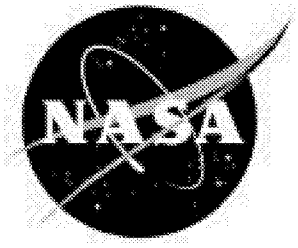
NASA/TM-2002-211632



Fly-By-Light/Power-By-Wire Fault-Tolerant Fiber-Optic Backplane

Mahyar R. Malekpour
Langley Research Center, Hampton, Virginia

April 2002

The NASA STI Program Office . . . in Profile

Since its founding, NASA has been dedicated to the advancement of aeronautics and space science. The NASA Scientific and Technical Information (STI) Program Office plays a key part in helping NASA maintain this important role.

The NASA STI Program Office is operated by Langley Research Center, the lead center for NASA's scientific and technical information. The NASA STI Program Office provides access to the NASA STI Database, the largest collection of aeronautical and space science STI in the world. The Program Office is also NASA's institutional mechanism for disseminating the results of its research and development activities. These results are published by NASA in the NASA STI Report Series, which includes the following report types:

- TECHNICAL PUBLICATION. Reports of completed research or a major significant phase of research that present the results of NASA programs and include extensive data or theoretical analysis. Includes compilations of significant scientific and technical data and information deemed to be of continuing reference value. NASA counterpart of peer-reviewed formal professional papers, but having less stringent limitations on manuscript length and extent of graphic presentations.

- TECHNICAL MEMORANDUM. Scientific and technical findings that are preliminary or of specialized interest, e.g., quick release reports, working papers, and bibliographies that contain minimal annotation. Does not contain extensive analysis.

- CONTRACTOR REPORT. Scientific and technical findings by NASA-sponsored contractors and grantees.

- CONFERENCE PUBLICATION. Collected papers from scientific and technical conferences, symposia, seminars, or other meetings sponsored or co-sponsored by NASA.

- SPECIAL PUBLICATION. Scientific, technical, or historical information from NASA programs, projects, and missions, often concerned with subjects having substantial public interest.

TECHNICAL TRANSLATION. English-language translations of foreign scientific and technical material pertinent to NASA's mission.

Specialized services that complement the STI Program Office's diverse offerings include creating custom thesauri, building customized databases, organizing and publishing research results . . . even providing videos.

For more information about the NASA STI Program Office, see the following:

- Access the NASA STI Program Home Page at ***http://www.sti.nasa.gov***

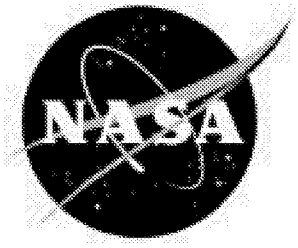
- Email your question via the Internet to help@sti.nasa.gov

- Fax your question to the NASA STI Help Desk at (301) 621-0134

- Telephone the NASA STI Help Desk at (301) 621-0390

- Write to:
  NASA STI Help Desk
  NASA Center for AeroSpace Information
  7121 Standard Drive
  Hanover, MD 21076-1320

NASA/TM-2002-211632



Fly-By-Light/Power-By-Wire Fault-Tolerant Fiber-Optic Backplane

Mahyar R. Malekpour
Langley Research Center, Hampton, Virginia

National Aeronautics and
Space Administration

Langley Research Center
Hampton, Virginia 23681-2199

April 2002

Available from:

NASA Center for AeroSpace Information (CASI)
7121 Standard Drive
Hanover, MD 21076-1320
(301) 621-0390

National Technical Information Service (NTIS)
5285 Port Royal Road
Springfield, VA 22161-2171
(703) 605-6000

Abstract

The design and development of a fault-tolerant fiber-optic backplane to demonstrate feasibility of such architecture is presented. The simulation results of test cases on the backplane in the advent of induced faults are presented, and the fault recovery capability of the architecture is demonstrated. The architecture was designed, developed, and implemented using the Very High Speed Integrated Circuits (VHSIC) Hardware Description Language (VHDL). The architecture was synthesized and implemented in hardware using Field Programmable Gate Arrays (FPGA) on multiple prototype boards.

Acknowledgments

I would like to acknowledge my gratitude to Dr. Jerry H. Tucker of NASA Langley Research Center for his guidance during the development process. I would like to acknowledge my appreciation to Dr. Celeste M. Belcastro of NASA Langley Research Center for her recommendations. I would also like to acknowledge my appreciation to Dr. Paul S. Miner of NASA Langley Research Center for his helpful comments in earlier version of this report. Lastly, I would like to acknowledge my appreciation to Wilfredo Torres-Pomales of NASA Langley Research Center for his review and helpful comments of the final version of this report.

Table of Contents

List of Figures

Acronyms

ASIC	Application Specific Integrated Circuit
BIU	Bus Interface Unit
RMU	Redundancy Management Unit
MUX	Multiplexer
VHSIC	Very High Speed Integrated Circuit
VHDL	VHSIC Hardware Description Language
FPGA	Field Programmable Gate Array
EPROM	Electrically Programmable Read Only Memory
RAM	Random Access Memory
FIFO	First In First Out
FBL/PBW	Fly-By-Light/Power-By-Wire
PLL	Phase Locked Loop (Analog)
DPLL	Digital Phase Locked Loop
Mbps	Mega bits per second

The use of brand names is for completeness and does not imply endorsement by US government.

1. Introduction

The purpose of this project is to develop an architecture capable of implementing the fault-tolerant, fiber-optic backplane proposed by Palumbo in [1]. The development of this architecture is also intended to assist with the investigations of behavior of the backplane in the presence of faults. The fiber-optic backplane consists of a set of Bus Interface Units (BIU) and Redundancy Management Units (RMU) forming multi-channel redundant fiber-optic backplane. Each channel, in turn, consists of a set of BIUs that are tied to a RMU via separate fiber-optic *read* and *write buses* (the action of read and write are taken from the perspective of the BIUs). Figure 1 is a depiction of the fiber-optic channel. Fault-tolerance is achieved by replicating several channels and combining the BIU outputs of the channels in the RMUs to mask any errors or failures before the data is placed on the *read buses*. In such redundant system, the RMUs of different channels communicate with each other through separate fiber-optic backplane *write buses*. Figure 2 is a depiction of the fiber-optic backplane. The RMUs also provide global time synchronization across the backplane and timing control through the channel *read buses*. All processing and bus accesses are controlled by time, i.e., all data appearing on the backplane can be uniquely identified by the time at which they become available. The BIUs of the channels are time division multiplexed onto the channel *write bus*. The RMU is the only device that writes to the channel *read bus*. Because the RMU is fundamental to the backplane's operation, both the channel *read bus* lines and the RMU may be replicated to increase reliability. Finally, the RMU can be integrated with a gateway to a network thus providing fault-tolerant access to remote processing nodes[1].

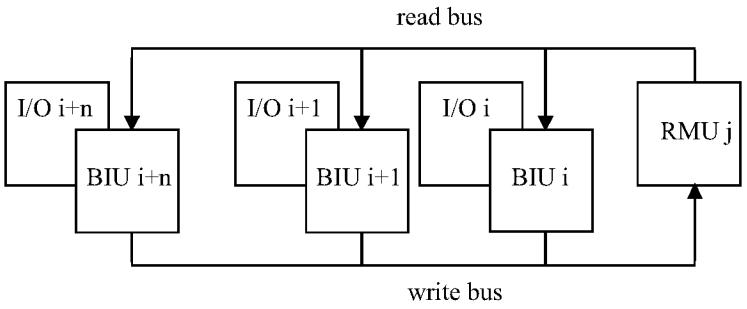


Figure 1. Fiber-Optic channel.

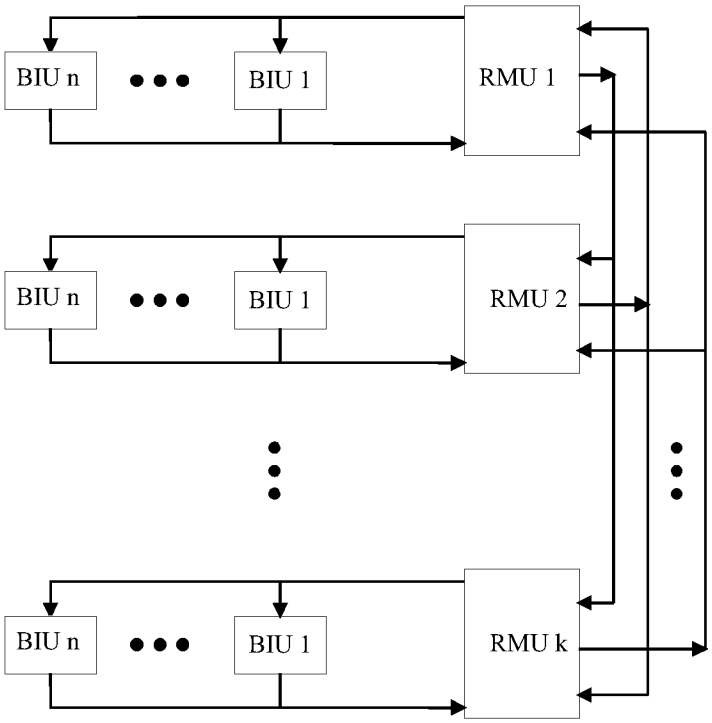


Figure 2. Fiber-Optic backplane.

Analysis of the backplane indicates that development of a single channel is sufficient for a feasibility study of the proposed backplane. Thus, the architecture developed, as shown in Figure 1, has been demonstrated with only one channel. In order to incorporate fault-tolerance into the system, additions required to accommodate multiple channels would have to be made to the RMU. The particular implementation of the architecture that is presented here enables a RMU to connect to as many as 29 BIUs; however, for testing purposes a maximum of four BIUs are sufficient to demonstrate full channel functionality.

The architecture is designed, developed, and implemented using the Very High Speed Integrated Circuits (VHSIC) Hardware Description Language (VHDL) [2]. Time constraints did not allow for a full hardware implementation; however, large portions of the developed architecture were synthesized and implemented in hardware using Xilinx Field Programmable Gate Arrays (FPGA) [3] on multiple prototype boards. These FPGA boards can be installed in Personal Computers (PC) such that the PCs act as the front-end to the FPGA boards for both programming the FPGAs and for controlling the operation and data transfer to the FPGA boards during their normal operations. Instead of designing one board to function as a RMU and designing a different board to function as a BIU, it was decided to take advantage of the flexibility provided by FPGAs to develop a single design so that a board could be programmed to function as either a RMU or a BIU.

This report presents the development and test cases of a single fiber-optic channel. In Section 2 the implementation issues, the design, and the development of the architecture are discussed. The hardware design and development of the architecture are presented in Section 3. Test cases and simulation results are presented in Section 4. Section 5 concludes this report with a summary of the work

accomplished and a discussion of future enhancements.

Five appendices supplement this report. Appendix A includes the VHDL code for the components of this architecture. Appendix B contains the C code. Appendix C describes the pin assignment and layout. Appendix D consists of sample schedules and data packets, and lastly, Appendix E describes the procedures for using all VHDL tools in the development process.

2. Design and Development

As stated in [1] "to support high speed data transmission, the optical receiver is clocked by a phase locked loop (PLL) which has locked its internal clock to the incoming data stream. Normally, switching between multiple data streams would represent a problem as this would require the PLL to re-lock. In this invention, the multiple transmitters in the BIUs, Figure 1, are themselves clocked by PLLs which are in turn locked to the data stream produced by the RMU transmitters. The multiple transmitters thus have the same clock source reducing skew and drift and minimizing lock time for the RMU PLL."

The proposed backplane requires fast PLLs with very low lock time. Specifically, the proposed design requires a PLL with a lock time of a few clock ticks while existing PLLs and DPLLs have a typical lock time of hundreds of ticks. Our investigations at the initial phase of the development process on existing PLLs and DPLLs revealed that existing commercial products did not meet the stringent requirements of the proposed design. The design of a new PLL or DPLL requires more study and is beyond the scope of this work. As a result, a new alternative is developed to 1) Meet the stringent timing requirements, 2) Allow continuation of the design and development of the architecture, and 3) Maintain interoperability with the backplane in the advent of new development in PLL technology, and 4) Keep the added cost to a minimum.

This alternative incorporates the use of a *Global Clock* over a separate fiber-optic cable, Figure 3. The *Global Clock* resides in the RMU and is broadcast to all BIUs in the channel. In this alternative, the BIUs are assumed to be at equal distances from the RMU of the channel. In other words, the *read* and *write* buses are of equal lengths. Therefore, all BIUs are guaranteed to be in perfect synchronization with the RMU and, as a result, the switch-time between the channels is at its absolute minimum of one clock tick. In addition, in the advent of new and fast PLL technology, the PLL output would simply replace the *Global Clock* input to the BIUs. The additional cost of this alternative is, therefore, associated with a transmitter, a fiber-optic cable, and receivers that are dedicated to the broadcast of the *Global Clock*.

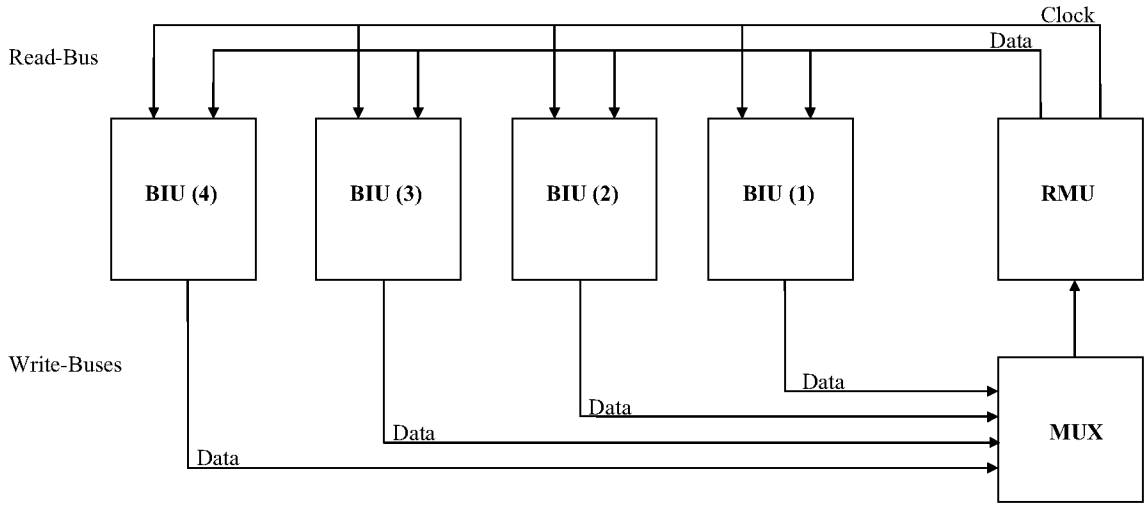


Figure 3. Global clock, fiber-optic channel.

2.1 BIU/RMU

Analysis of the behavior of the RMU and BIU revealed that these modules have so much in common that the BIU should be treated as a special case of the RMU, Figure 4. In particular, the main functions of the BIU and RMU are transmission of data, reception of data, and execution of the scheduled instructions. Of course, RMU interpretation of the scheduled operations is slightly different from the BIU. The only RMU-specific function is voting on the input data and masking out the faulty BIU(s). However, this function may be performed by an independent module that complements the BIU module's functionality. As a result, both BIU and RMU can be designed to have identical interfaces to the outside world. Therefore, the terms BIU and RMU are used interchangeably in the implementation sense. However, every instance of this module requires its own unique identifier. This identifier is set externally via the BIU_ID parameter. Also, by accommodating for their differences in interpreting the scheduled operations via an external bit (*BIU_OR_RMU*), the BIU/RMU architecture can be developed as a single module. Joint development of the BIU/RMU has the added advantages of requiring less development time and code maintenance. Also, it reduces the overall ASIC fabrication cost by 50% since one single die suffices. Therefore, for the remainder of this report, unless specifically stated, all details and descriptions of this module apply to both BIU and RMU. The VHDL entity declaration and architectural description of the BIU/RMU are listed in Appendix A.

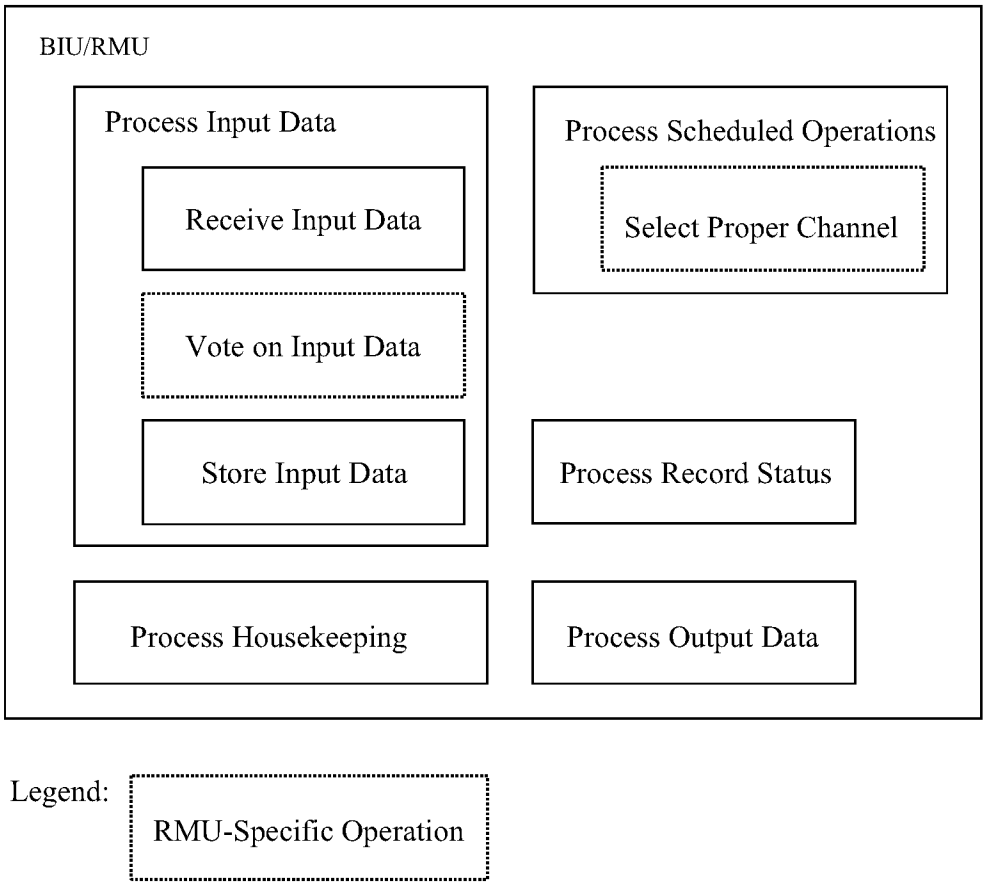


Figure 4. BIU/RMU functional descriptions.

The Process Housekeeping module handles the power on and reset conditions by initializing the internal counters, registers, and resetting the local timers. The details of this process are described in the Housekeeping section.

The Process Input Data module continuously monitors the incoming data by converting the bit-serial data stream to parallel words. It then stores the incoming data packet in the appropriate buffers to be used by the rest of the system. The details of this process are described in the Input Data section.

The Process Output Data module transmits the outgoing data at the specified scheduled time. The output data are either internal status report from the BIU or output data of the BIU's associated processor. Regardless, the output data words are first packetized with the appropriate header and then serialized for transmission. The details of this process are described in the Output Data section.

The Process Record Status module keeps track of the errors by setting their designated bits in the status register. The details of this process are described in the Reporting Errors section.

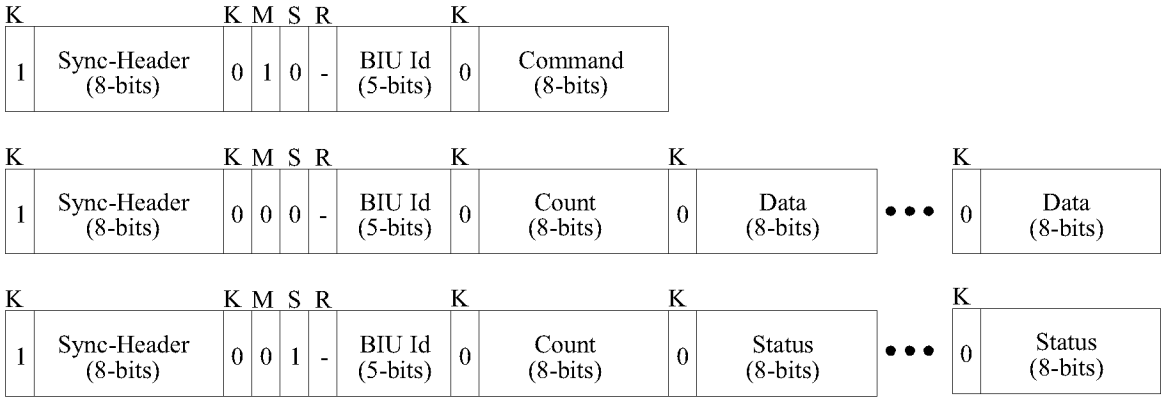
The Process Schedule Operation module manages loading of the scheduled operations from the ERPOM/RAM by setting the appropriate address lines and issuing the read signal. It then decodes the instructions and stores them in the appropriate buffers. The details of this process are described in the

Schedule Format and Schedule Controller sections.

The BIU/RMU has two types of interfaces: one to communicate with the BIU/RMU of the channel and the other to communicate with BIU's associated processor, Figure 1. The Input and Output modules are designed to communicate with the BIU/RMU as well as the associated processor. Although the BIU and RMU exchange data via serial fiber-optic buses, the data to and from the associated processor of the BIU are exchanged in parallel words using separate FIFOs. In order for this module to transmit and receive data simultaneously, two FIFO interfaces are, therefore, necessary to handle the input and output data flux.

2.2 Packet Formats

The data and status information as well as the commands issued by the RMU are stored in packets based on one of the formats depicted in the following figure. However, the type of packet format is based on the nature of the information to be sent to the destination BIUs.



Sync-Header, 8 bits = 1111_1111
Sync-Pattern, 10 bits = 1_1111_1111_0

BIU Id, 5 bits, identifies the destination of the packet.
Reserved = 0_0000
Global Id, Reserved = 1_1111

K = Sync Bit = 1 ==> Sync-Header follows
= 0 ==> Data, Command, or Status follows

M = Mode Bit = 1 ==> Command
= 0 ==> Data or Status

S = Status Bit = 1 ==> Status
= 0 ==> Data

R = Reserved Bit

Figure 5. Packet formats.

All packet formats share a common scheme. This underlying scheme consists of three 9-bit words where each word is constructed from an 8-bit byte that is preceded with a *synchronization bit (K).* The *synchronization bit* is zero except when indicating the *Sync_Header.*

The first word of a packet is the *Sync_Pattern*, the second word is a collection of flags and BIU/RMU identification, and the third word is either a command or a count. To achieve synchronization over a distance between BIUs and RMU, the *Sync_Pattern (1_1111_1111_0)* is designed so that it is guaranteed to be unique throughout the system. The *Sync_Pattern* is a unique 10-bit pattern consisting of a string of 9 ones followed by a zero. Since the first bit of the second word has to be a zero, that bit is used as part of the *Sync_Pattern.*

The second word consists of three 1-bit flags; *Mode (M)*, *Status (S)*, and *Reserved (R)* flags, followed by a 5-bit identification field that is used for both BIU and RMU. The significance of the third

word depends on the flags that are set. If the *Mode* (M) bit is set, then the third word is a command for the BIUs; otherwise, it is a count of data or status words to follow. In this case, the packet will be more than three words long. Since *count* is an 8-bit field, the maximum number of status or data is limited to 255 words per packet.

If the *Status* (S) bit is set, then the packet holds status information and thus it is forwarded to the output FIFO. Otherwise, it is a data packet intended for the BIU whose identification is in the packet header. In this case, only the target BIUs fetch the packet and forward it to their associated processors (via the output FIFOs), while all other BIUs simply ignore the packet.

The *BIU Id* can be used as another layer of redundancy to check against scheduled operations for local detection of failures.

The *Reserved* (R) bit is not used at this time. It could be used as part of the BIU/RMU identification and to expand the number of BIUs in a single backplane channel.

2.3 Performance

The calculation of the *read bus* bandwidth efficiency as a function of packet size follows.

Packet Overhead = MUX Switch Time + Data Packet Header + Overhead per Data Byte

where,

MUX Switch Time = 1 clock tick = 1 Word = 9 bits

Data Packet Header = (Sync_Header + Flags) + BIU_Id + Count of Data Words

= 3 Words = (3 * 9) bits

Overhead per Data = 1 bit

So, with n = Number of Data Words as specified in Count Field,

Overhead = 9 + (3 * 9) + (1 * n) = 4 * 9 + n = (36 + n) bits

%Overhead = Overhead / Packet Size * 100 = (36 + n) bits / ((4 + n) * 9) bits * 100

and

%Efficiency = 1- %Overhead = n * 8 / ((n + 4) * 9)

As evident from the above equation, as n grows, so does the %Efficiency. The *read bus* bandwidth efficiency is displayed as a function of data bytes in a packet in the following figure. As is shown, the efficiency approaches the maximum (about 89%) for moderate size packets.

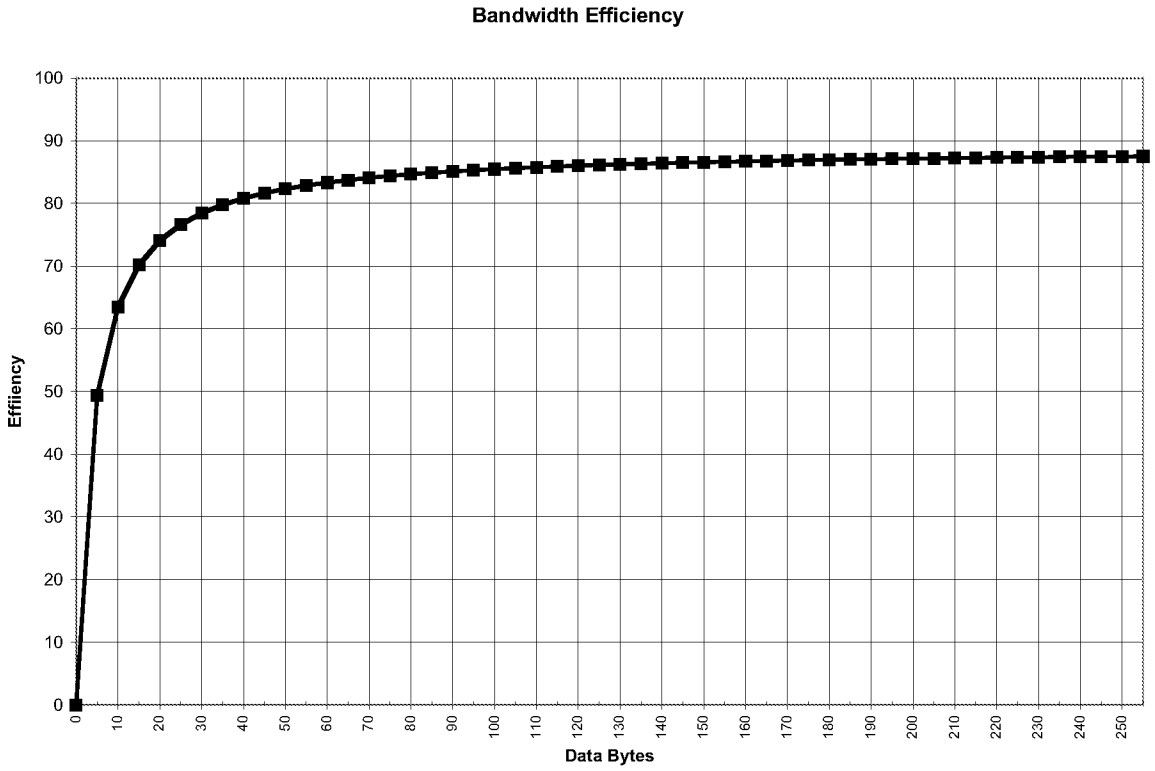


Figure 6. Channel *read bus* bandwidth efficiency as a function of packet size.

2.4 Reporting Errors

The status register, *Status_Reg_0*, is introduced to keep track of errors at various sub-modules. Figure 7 provides a detailed description of the status register. Various bits of this register indicate specific errors and, therefore, are set by their designated sub-modules upon detection of errors. The content of this register is transmitted at the scheduled times and after setting the *Status* (S) bit of the packet.

Bit	Error Name	Error Description
0	Read_FIFO_Error_1	Error in input FIFO data packet header
1	Read_FIFO_Error_2	Attempted to read from empty input FIFO, i.e. missing data
2	Receive_Error_1	Data didn't arrive within the expected reception window
3	Receive_Error_2	Received unexpected data
4	EPROM_Error_Flag	Didn't detect end-of-schedule in the EPROM/RAM
5	None	None
6	None	None
7	None	None

Figure 7. Status_Reg_0, error bit assignments.

2.5 Housekeeping

After power on and upon reset the BIU/RMU resets its internal counters, clears its registers, and resets its transmitter and receiver clocks. Figure 8 depicts the flowchart of the power on and reset activities. If the *BIU_OR_RMU* bit is set high, the architecture is that of a BIU. It goes into a *wait state* where the BIU awaits the *Start_Cycle* command from the RMU. Otherwise, the architecture is a RMU and begins reading the scheduled operations and takes appropriate actions at the right times. As further described in Section 3.8, Schedule Format, the first two instructions of the schedule are reserved for the RMUs only. The first instruction indicates broadcasting of the *Start_Cycle* command to all BIUs in the channel while the second instruction is a wait instruction for the RMU for the specified delta time so that the BIUs can catch up with the RMU. Upon receiving the *Start_Cycle* command, the BIU resets its internal counters, clears its registers, and resets its transmitter and receiver clocks. At this time, all BIUs are synchronized with respect to the RMU. The BIUs and RMU then repeat reading the scheduled operations and execute scheduled instructions at the specified times.

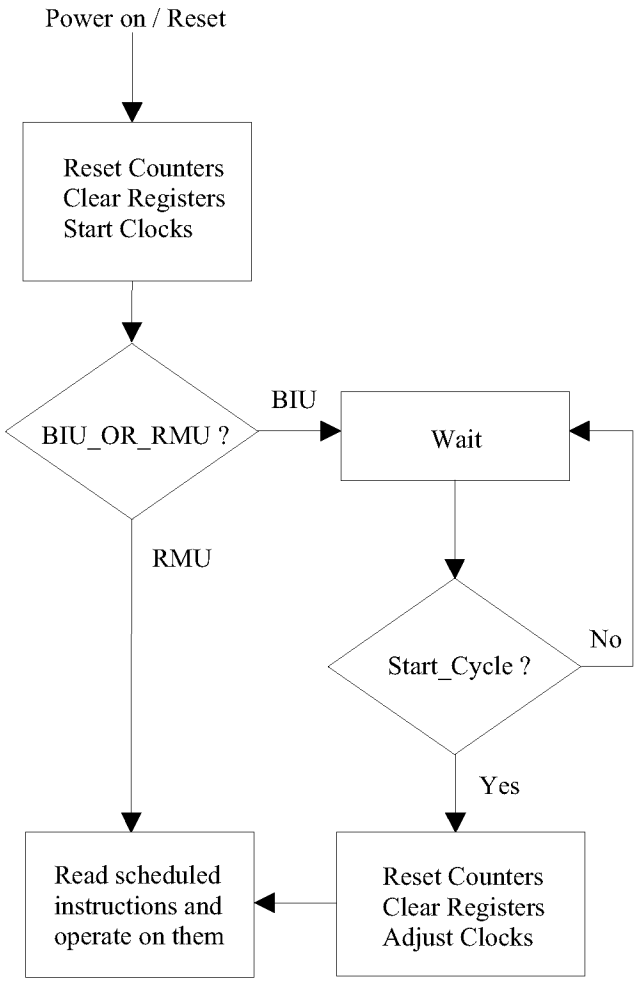


Figure 8. Power on/reset operations.

2.6 Input Data

Data reception requires continuous conversion of bit-serial data stream to parallel bytes. The incoming data bit stream is monitored to detect the *Sync_Pattern*. Figure 9 depicts the flowchart of the incoming-data controller. Upon detection of the *Sync_Pattern* the receiver clock is adjusted so that the following data are retrieved at appropriate word boundaries (see Section 3.6 System Clocks). If the BIU_ID part of the second word does not match the BIU_ID of the particular instance of this module, then the *Status bit* is examined. If the *Status bit* is not set then the rest of the data packet will be ignored. Otherwise, the packet is treated as a *Status packet* and is simply routed to a FIFO in its entirety. If the second byte matches the BIU_ID of a particular instance of this module, then the *Mode bit* is examined. If the *Mode bit* is not set then the rest of the data packet will be treated as a *data packet* for this module and will be routed to a FIFO. Otherwise, the packet is treated as a *Command packet* from the RMU and the proper action will be taken.

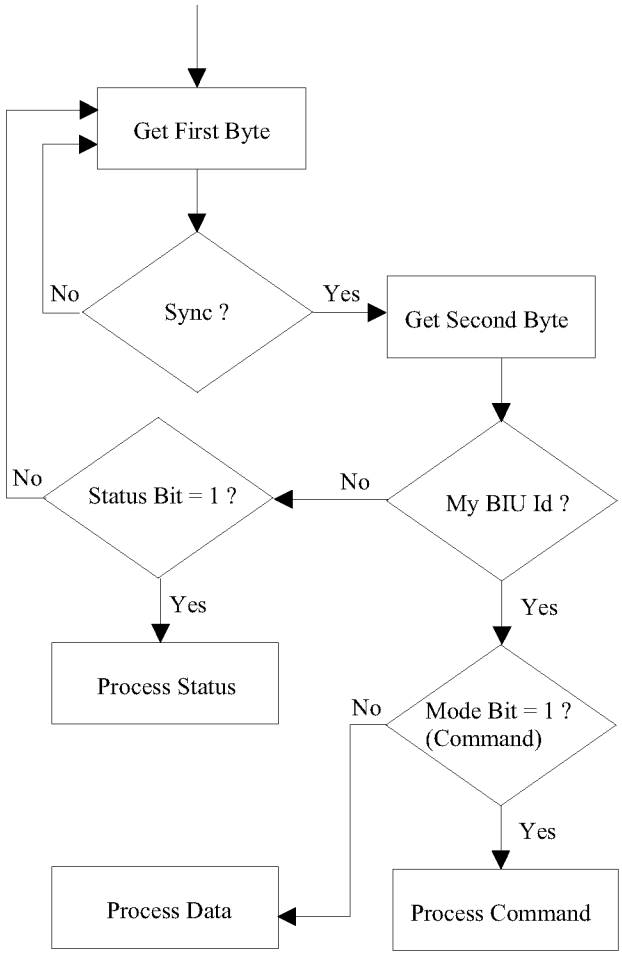


Figure 9. Incoming-packet controller.

To accommodate for minor variations in the lengths of the busses, a reception window is established to provide an added flexibility to the architecture. The duration of this window is controlled externally and can range from 0 to 7 byte clock ticks by setting the three *Switch_Time_In* bits. The maximum reception window of 7 byte clock ticks allows for a maximum of 7 bytes * 9 bits per byte * 10 ns per bit = 630 ns = 630 feet variations in the lengths of the busses (assuming a 100 MHz clock, and that light travels 1 ft/ns). The reception window starts at the scheduled data reception time, and lasts as long as the reception window size or until a data packet is received. If the scheduled data packet is not received during this time or if it arrives outside this window, then the errors are reported by setting their designated bits, bits 2 and 3, respectively, in the error register *Status_Reg_0*. Figure 7 provides a detailed description of the status register.

Voting of the data is an RMU-specific function and is performed by all RMUs in a redundant multi-channel system to provide fault tolerance for the full FBL/PBW backplane. In a redundant multi-channel system, all RMUs broadcast their input data to all other RMUs as data become available. Therefore, a BIU output is available to all RMUs at the same time. Each RMU then votes on the data it receives from RMUs of other channels and on the data from the corresponding BIUs of its channel, Figures 1 and 2. In case of any discrepancy, the faulty BIU is identified and masked out. The voted BIU

output is broadcast in the local channels. Since the design and development of the voter module is beyond the scope of this work, the voter implementation is left for future work.

2.7 Output Data

Data transmission requires reading a data packet from a FIFO, checking the data integrity by examining the packet header, and converting the data bytes into a continuous serial bit stream. Figure 10 depicts the flowchart of the outgoing-data controller. If the packet header, specifically the *Sync_Pattern,* is not detected at the expected time, then an error is registered and the transmission operation is aborted. Also, to avoid issuing any commands by the microprocessor to the RMU and to safeguard against any undesirable side effects, the *Mode bit* is examined. As previously described in Figure 5, if the *Mode* (M) bit is set, then that word is a command for the BIUs. Therefore, to guarantee that the commands are issued and, thus, the *Mode bit* is managed from inside the FBL/PBW backplane architecture (specifically only by the RMU), the *Mode bit* is examined and if it is set by the BIU's associated processor, then an error is registered and transmission operation is aborted.

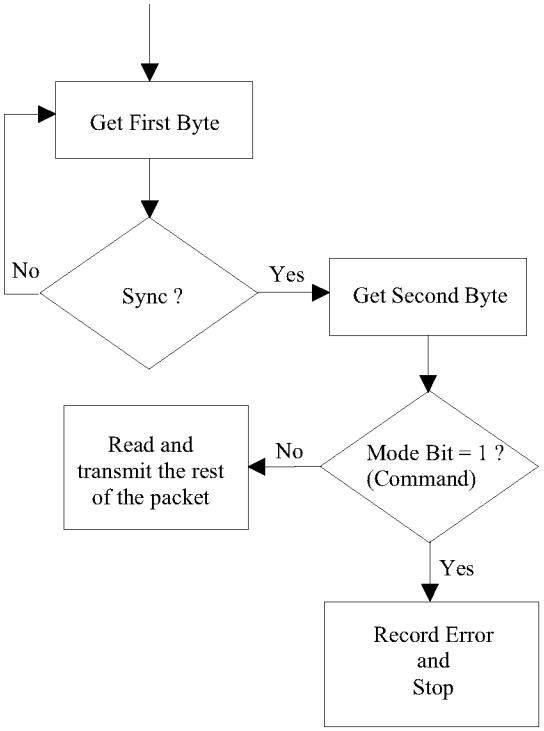


Figure 10. Outgoing-packet controller.

2.8 System Timers

To synchronize and maintain synchronization between the receiver of a BIU/RMU and the transmitter of another BIU/RMU at the proper word boundaries, the receiver needs to constantly adjust to the transmitter. As a result, the receiver part of a BIU/RMU must operate with a different timer than the

rest of the unit. To prevent propagation of phase shifts in the receiver timer to the rest of the system and safeguard against any side effects, a second timer, the transmitter timer, is introduced. Therefore, the BIU/RMU has two timer regions: a receiver timer region and a transmitter timer region. To maintain design flexibility, provisions are made so that the transmitter timer can be adjusted and synchronized with the receiver timer; however, it must only be done when the BIU/RMUs are in the idle state. The synchronization of the transmitter timer with the receiver timer is achieved as a scheduled event and at the desired synchronization interval via a RMU command.

Separation of the two timers has the added advantage of applicability to a broader class of architectures by eliminating the fix distance constraint between the BIUs and RMU of one channel as well as between the RMUs of multiple channels. In addition, the cost of the second timer, four flip-flops, is negligible.

Although the data in and out of the BIU/RMU are serial bit streams, the BIU/RMU operates at 9 bit word boundaries. Therefore, the BIU/RMU requires system timers that operate at the word level. Since the serial data bit streams are 9-bit words, the system clocks are derived from the incoming bit clock by dividing the bit clock by 9. Operating at the word level has the advantage that most of the BIU/RMU operates at a slower clock rate and the peripherals such as the FIFOs and EPROM/RAM can be slower devices. This slower clock rate allows for less stringent requirements on the signal load and routing, and therefore, is more cost effective. In addition and from the user's perspective, delta time for the scheduled operations will be with respect to the system timers and, hence independent of the communication rate.

2.9 FIFO

In order to make the simulation results comparable to those of the prototype boards, the generic FIFO module developed for this design is modeled after Am7204A[1] FIFO chips. This VHDL model is comparable with the Am7204A FIFO in both interface and timing characteristics. In addition, this VHDL FIFO model is a generic model so that by adjusting its parameters, it can be defined to be as wide, deep, and fast as necessary. This VHDL model is synthesizable and the VHDL code is included in Appendix A.

2.10 EPROM

For simulation purposes a high level VHDL model of a generic EPROM was developed that is pin-to-pin and package compatible with a generic RAM. However, for design flexibility, the interface for this module is modeled after the NM27C128, 128k-word x 8-bit EPROM[2] and HM6264ALSP, 8192-word x 8-bit High Speed Static CMOS RAM[3] which are pin-to-pin and package compatible. The EPROM module contains the scheduled instructions and the relative time of their operations. The EPROM has to be 16-bit wide and deep enough to hold all scheduled events. The schedule format is described in the following section and schedule examples are listed in Appendix D.

---

[1] Am7204A is a CMOS FIFO and a product of the Advanced Micro Devices.
[2] Products of National Semiconductor Corporation.
[3] Products of HITACHI Corporation.

2.11 Schedule Format

The scheduled events and instructions are stored in an EPROM or a RAM based on the format depicted in the following figure.

Delta Time (8-bits)	Transmit (1-bit)	Receive (1-bit)	Status/Data/Command (1-bit)	BIU/RMU Id (5-bits)

EPROM/RAM Width = 16 bits
Status/Data/Command = S/D/C

ΔT =	Delta Time	>= 0 ==> Delta Time between consecutive instructions
Tx =	Transmit Bit	= 1 ==> Transmit
		= 0 ==> No-op
Rx =	Receive Bit	= 1 ==> Receive
		= 0 ==> No-op
S/D/C =	Status Bit	= 1 ==> Data for BIU and Command for RMU
		= 0 ==> Status
Id =	BIU Id	= 1 .. 30 (base 10)

End of schedule delimiter is 31 in base 10 (i.e. XXFF in base 16)

Tx	Rx	S/D/C	Descriptions
0	0	0	RMU and BIU No-op
0	0	1	N/A
0	1	0	N/A
0	1	1	RMU and BIU Receive Data
1	0	0	RMU and BIU Transmit Status
1	0	1	RMU Transmit Command and BIU Transmit Data
1	1	0	N/A
1	1	1	RMU and BIU Stop

Example:

In the following schedule example, RMU Id = 27 and Global Id = 31.

ΔT	Tx	Rx	S/D/C	Id	Descriptions
10	1	0	1	27	RMU will transmit Start_Cycle command after 10 clock cycles
5	0	0	0	27	RMU will do nothing and waits for 5 clock cycles until BIUs restart
.....					
5	1	0	0	27	RMU will transmit its status after 5 clock cycles
1	0	1	1	31	All BIUs should expect to receive data in 1 clock cycle
4	1	0	0	1	BIU 1 will transmit its status information after 4 clock cycles
2	0	1	1	31	All BIUs should expect to receive data after 2 clock cycles
.....					
19	1	0	1	3	BIU 3 will transmit its data after 19 clock cycles
1	0	1	1	4	BIU 4 should expect to receive data after 1 clock cycle
.....					
15	1	1	1	31	i.e., 0FFF, all BIUs stop reading the schedule after 15 clock cycles

Figure 11. Schedule format for EPROM/RAM.

The scheduled events are 16 bits wide, i.e. two 8-bit bytes. The little-endian notation is used to describe different segments of the schedule events. The first byte is reserved for delta time. This allows for a time interval between two consecutive events to be at most 256 system timer ticks. However, to extend this time interval beyond 256 clock ticks, no-op instructions should be inserted between the actual events. The three most significant bits of the second byte are used in the communication process. Specifically, bit 7 of the second byte indicates transmission event, bit 6 indicates receiving event, and bit 5 indicates the nature of the event as being status, data, or command. The five least significant bits, bits 4 through 0, identify the RMU/BIU that is scheduled to take the appropriate action after the delta time has elapsed. Therefore, this format allows for one RMU and a maximum of 29 BIUs per channel.

The first two instructions of the schedule are reserved for the RMUs only. The first instruction indicates broadcasting of the *Start_Cycle* command to all BIUs in the channel. The second instruction is a wait instruction for the RMU for the specified delta time so that the BIUs can catch up with the RMU. The duration of the wait time is a function of the communication means and the delay in processing of the *Start_Cycle* command by the RMU and BIUs. The wait time, therefore, is given by the following equation:

Wait Time = Command Process Delay + Read Bus Delay

The *Command Process Delay* is a constant delay and is determined to be five system timer ticks for this implementation. It is the total delay in constructing the package, transmitting the *Start_Cycle* command by the RMU, and receiving the command by the BIUs. The *Read Bus Delay* is determined by the time it takes for the data to reach from the RMU to the BIUs of the channel and is directly proportional to the length of the bus. Since the BIUs are assumed to be of equal distances from the RMU of the channel, after elapse of the wait time, the BIUs will be synchronized with respect to the RMU. The example depicted in Figure 11 indicates a *Delta Time* of 5 system timer ticks. The second instruction of the schedule corresponds to the *read bus* delay of zero.

When the bit 7 of the second byte is set high, it is interpreted by the BIUs as a transmit instruction. However, the RMU interprets it as a switch channel instruction and uses the BIU identity field, bits 4 through 0, as the multiplexer select lines to switch to the appropriate BIU *write bus*.

2.12 Schedule Controller

Reading of the scheduled operations from the EPROM/RAM requires setting the appropriate address lines and issuing the read signal. The Schedule Controller manages loading of the instructions from the EPROM/RAM. The scheduled instructions are pre-fetched, decoded, and stored in appropriate buffers. In particular, the time field is extracted and stored in the *Delta_Time_Clock* and the instruction field in the *Instruction_Buffer* registers. The current instruction is then decoded. The corresponding flags that initiate the execution of the specific operations, such as transmit and receive, are raised only after the elapse of the delta time. Section 3.9 provides a detailed description of the scheduled instructions in EPROM/RAM.

2.13 BIU Testbench

The BIU testbench, Figure 12, encompasses the BIU/RMU and all the necessary components for its normal operations as a separate prototype board. The adjoining components are an EPROM/RAM that contains the scheduled events of operations, a FIFO for the input data, a FIFO for the output data, and a microprocessor (PC) with its associated input and output files that acts as the BIU font-end. A single external bit (*BIU_OR_RMU* = VCC) specifies its functionality to be a BIU. These components are an integral part of testing BIU/RMU functionality.

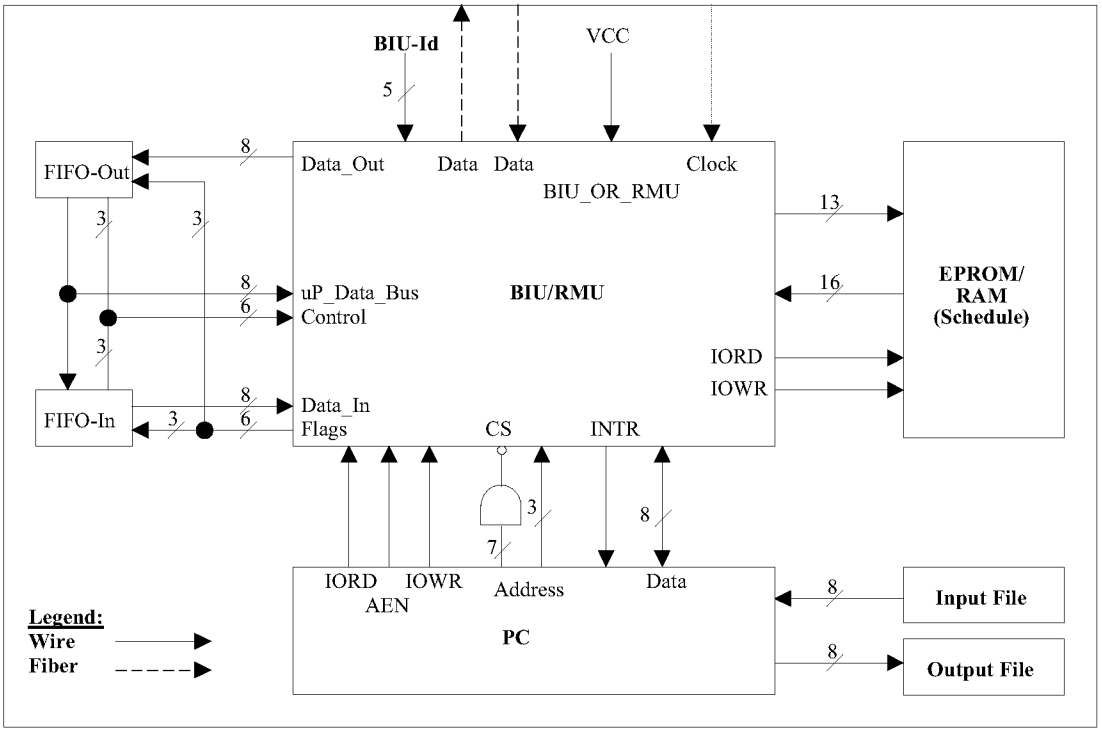


Figure 12. BIU testbench.

2.14 RMU Testbench

Analysis of the behavior of the RMU revealed that by preserving the BIU interface to the FIFOs, the RMU's interface could be defined as a special case of the BIU's interface. As a result, one FIFO is used for both input and output of data for the RMU.

The RMU testbench, Figure 13, encompasses the BIU/RMU and all the necessary components for its normal operations as a separate prototype board. The adjoining components are an EPROM/RAM that contains the scheduled events of operations, a single FIFO for both the input and output data, and a microprocessor (PC) with its associated input and output files that acts as the RMU front-end. The RMU testbench, therefore, is similar to the BIU testbench and by proper setting of a single external bit (*BIU_OR_RMU* = GND), its functionality is distinguished from that of BIU. These components are an

integral part of testing BIU/RMU functionality.

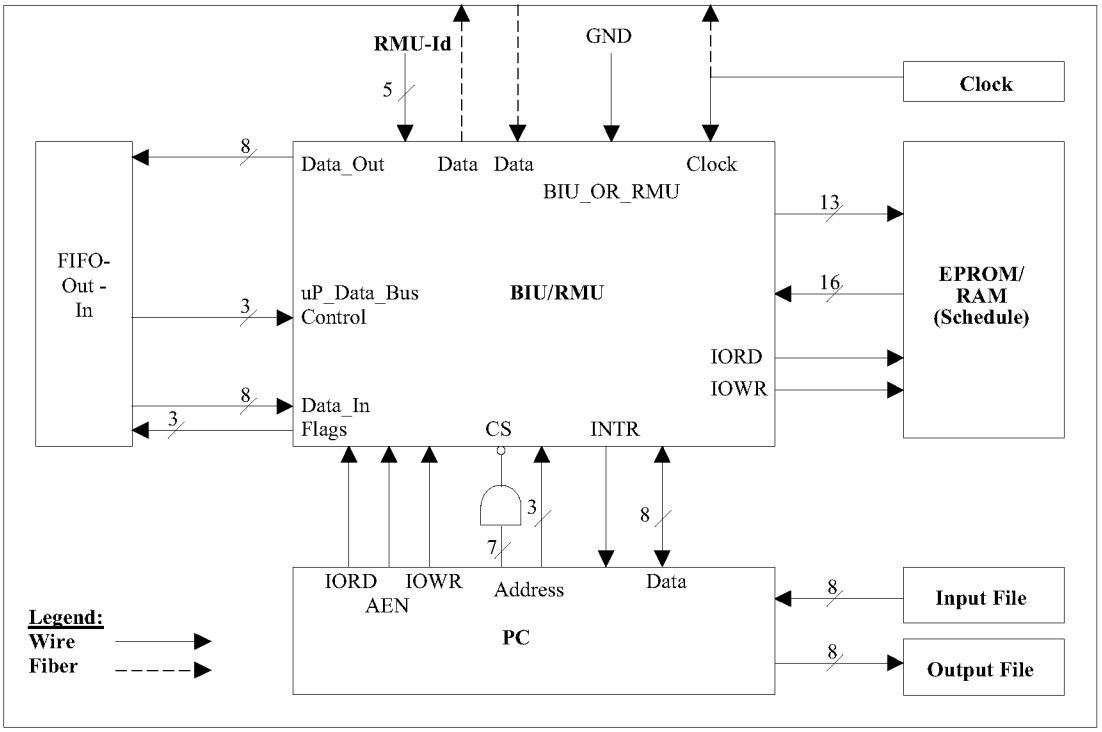


Figure 13. RMU testbench.

2.15 Microprocessor (PC)

The microprocessor (PC) is a high level representation of a generic microprocessor and is designed for simulation and testing purposes only. For simulation and testing flexibility, the microprocessor is designed so that it could be tailored to represent processors with different read cycles, different write cycles, and different clock rates. Also, different instances of this module can be programmed to transmit different counts of data packets with different data packet sizes. However, a particular instance of a microprocessor transmits a given number of data packets of the same sizes. Also, the microprocessor is assumed to have an identical copy of the BIU/RMU schedule.

The microprocessor operations are shown in the following flowchart, Figure 14. Since the microprocessor is assumed to be independent of the BIU, the communication between the BIU and its associated microprocessor is therefore asynchronous. As a result, the microprocessor receives an interrupt from its associated BIU at the start of every schedule cycle and after receiving the *Start_Cycle* command from the RMU. The microprocessor will reset the FIFOs, sample the sensors, send the data to the input FIFO, and then acknowledges the interrupt to the BIU. Note that the microprocessor can read processed data from the output FIFO at any time. This data is assumed to be stored in either a large cache or an output file. For simulation and testing purposes the data sent to the input FIFO is the output of counters internal to the microprocessor module.

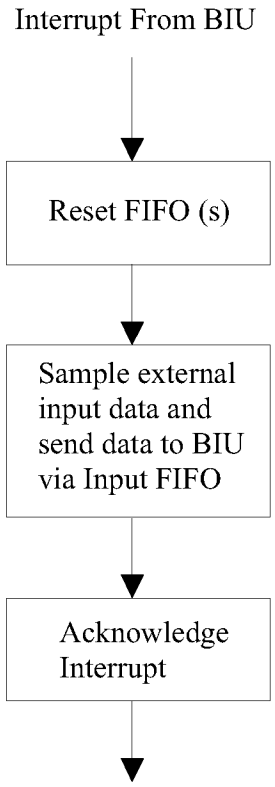


Figure 14. Microprocessor operations.

2.16 Fault Injection

There are many methods of injecting faults in the system. Three methods of injecting faults into this system are described here. The first is the brute force method where a BIU is turned off. Since at power down the exact state and condition of the BIU is not known, this method of fault injection is random. In simulation, however, turning off the BIU can be accomplished by forcing the BIU to reset where it waits in the *idle state* during the simulation process. This method of fault injection covers the fail silent scenario.

The second way of injecting a fault is through the schedule and by instructing the BIU to stop transmitting data at a specific time. In effect, the BIU goes off line at the designated time. As a result, the time of fault occurrence is predicable. Since the fault can be scheduled to occur at a specific time, this method is extremely helpful in examining integrity of the system in the presence of a fault at different states of the system. This method, therefore, provides a general means to analyze the architecture under various crash failures.

The third method is also through the schedule but by switching the channel to another BIU, preferably an unattached BIU. As a result, even though all BIUs are functioning normally, switching to a bogus channel will in effect disrupt proper routing of the intended BIU output to the target BIUs. This method can simulate data packet corruption through the *write bus* as well as BIU babbling.

These fault injection methods cover only a subset of the faults that this architecture is designed to tolerate. In the interest of time, further failure analysis and evaluation of this architecture is left for future work.

2.17 Fault Recovery

In the case of brute force method where a BIU is powered down, the BIU can be reintroduced into the system upon power on and at the start of the next schedule cycle. At power on, the BIU resets its internal registers and enters the *idle state*, Figure 8, awaiting the *Start_Cycle* command from the RMU before restarting its normal operations. Therefore, this fault recovery capability lends itself to upgrading the system by taking the BIUs off line, one at a time, and without having to power down the whole system.

In all other cases, where a BIU is either babbling or is not transmitting data, the BIU may recover from the fault provided that the fault is not persistent. In that case, the BIU may recover at the start of the next schedule cycle and upon receiving the *Start_Cycle* command from the RMU. However, if the fault persists for more than one schedule cycle, then the BIU may never recover.

2.18 Reporting Faults

Regardless of the nature and timing of the faults, as far as the rest of the system is concerned, the symptoms are the same. These symptoms eventually show up on the *read* and *write* busses. When matched against the scheduled activities on these busses, the faulty BIU and nature of the fault is identified. The symptoms indicate whether the faulty BIU is babbling or is not transmitting at the scheduled time. These errors are reported by setting their designated bits, bits 2 and 3, respectively, in the status register *Status_Reg_0*. Figure 7 provides a detailed description of the status register. A more descriptive error reporting would require time stamping the errors. However, this implementation is left to future enhancements.

3. Hardware Development

The FBL/PBW backplane was developed using VHDL. The VHDL code was synthesized using Synergy, a Cadence product, and targeted for the Xilinx FPGAs [3]. The FPGAs along with other off-the-shelf ASIC devices were used to construct a prototype board that would plug into the PC-AT bus. The PC was then used as the front-end to the prototype boards for both programming the FPGAs and for controlling the operations and data transfer to the boards during the normal operations. Instead of designing one board to function as a RMU and designing a different board to function as a BIU, it was decided to take advantage of the flexibility provided by FPGAs to develop a single design so that a board could be programmed to function as either a RMU or a BIU. To be able to program the FPGAs, a PAL was used to decode the base address of the I/O ports on the prototype board. For design flexibility, a generic interface was designed so that any microprocessor could interface with the board during its normal operations. This generic interface was separately programmed on a XC3020 [3]. The rest of the VHDL code encompasses the BIU/RMU module and was programmed on a XC4005A [3] that was selected for its size and larger number of I/O pins than the XC3020. The prototype board was wire-wrapped, tested, and its functionality verified. The prototype board functioned at 40 Mbps and demonstrates that the FBL/PBW backplane implementation was feasible.

3.1 PAL

The PC interface logic[4] for programming the XC3020 of the prototype board was implemented using a PALL22V10[5]. The base address of the prototype board was 300H. Address 306H was used to reset and address 307H was used to reprogram the XC3020 FPGA. The rest of the addresses were used to interact with the FIFOs and XC4005A. When resetting the XC3020, bits 0 and 1 of the PC data bus were used to control the RESET and Done/Program signals of the XC3020, respectively. For programming of the XC3020 the PC data bus bit-0 was used to download the binary file to the XC3020. The VHDL implementation of this interface is listed in Appendix A, the related C code is listed in Appendix B, and the pin assignment is listed in Appendix C.

[4] IBM, "IBM Technical Reference for Personal Computer AT, # 6280070."
[5] Product of AMD Corporation.

3.2 Address Assignment

The address assignment and their purposes in the prototype board are as follows:

New Address	Device	Function
300H	XC3020	Read/Write FIFOs
301H	XC3020	Read status of FIFOs
302H	XC3020	Write status (Reset FIFOs)
303H	XC4000	Transfer Data
304H	None	None
305H	XC3020	Reset and Program XC4000
306H	PAL	Reset XC3020
307H	PAL	Program XC3020

3.3 XC3020 and Microprocessor Interface

The XC3020 was programmed with a generic interface to allow a BIU and its associated application microprocessor to exchange data. The application microprocessor is assumed to be either an Intel 80X86 type or Motorola 68XXX type. In the prototype board the host PC played the role of application microprocessor after initial board setup and programming of the FPGAs. Since the application microprocessor accesses the FIFOs through its data bus and performs either read or write operation, the output bus of the output FIFO and the input bus of the input FIFO are tied to the microprocessor data bus via a bus controller. When exchanging data with the FIFOs, the bus controller relinquishes control to the microprocessor; otherwise, it tri-states the bus so that there will not be any interference with the microprocessor's normal operations. The VHDL implementation of this interface is listed in Appendix A and the related C code is re listed in Appendix B.

3.4 Programming XC4005A and Testing FIFOs

Upon setting up the prototype board and programming the XC3020, the XC4005A can be reprogrammed to implement the BIU/RMU functionality. Independent programming of the XC3020 and XC4005A allows for ease of modification to the BIU/RMU without having to turn off and on the PC and setting up the prototype board. The control signals of the XC4005A, i.e., *Program* and *Done* signals, are brought into the XC3020, and the XC4005A status are stored in a status register. Contents of this register are then accessed by the microprocessor for test and debugging purposes. Also, to enable monitoring of the status of the FIFOs, the FIFO status flags, e.g., *Full-Flag*, *Empty-Flag*, and *Half-Full*, are also stored in a status register and are accessed by the microprocessor. The VHDL implementation of this interface is listed in Appendix A and the related C code is listed in Appendix B.

4. Simulation Results and Test Cases

In this section two test cases are presented to demonstrate the capabilities of the FBL/PBW backplane. In the first test case the system operation under ideal conditions is examined. In the second case failure of a BIU due to power down or reset is studied.

The single channel under study consists of one RMU and four BIUs. To examine the operations of the system under various conditions, a generic schedule is setup to encompass all aspects of the fault injection and recovery while exercising all BIUs. In these test cases the schedule consists of transmission windows for the BIUs in the following order: 1, 2, 3, 4, 1, 2, 1, and 3. The following figure shows the typical activities of the BIUs during one scheduled period in the absence of faults.

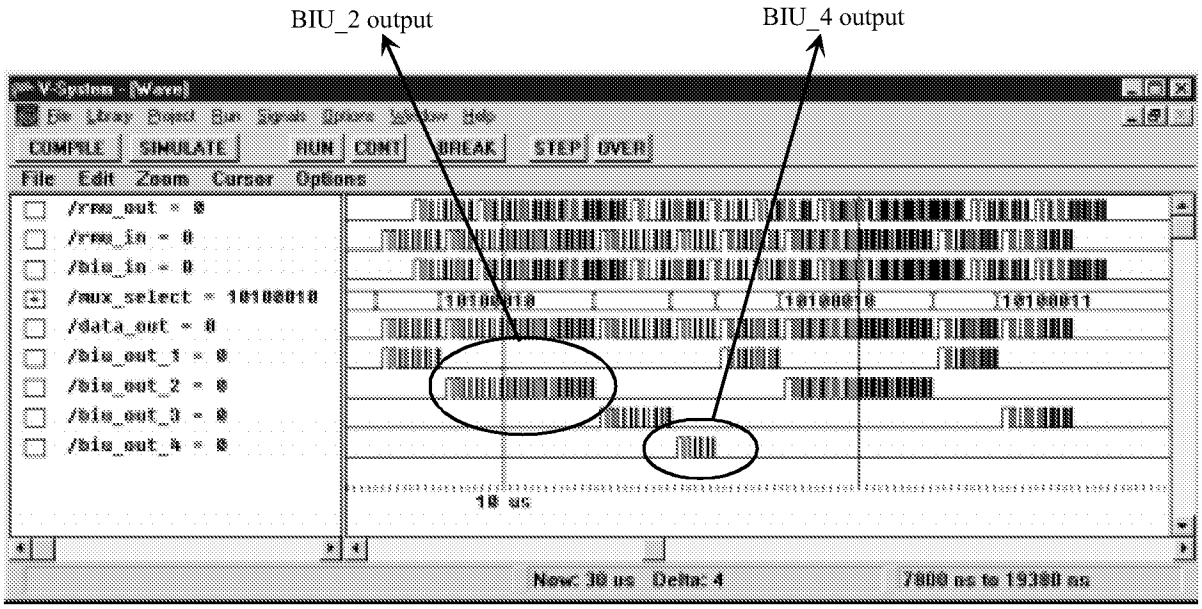
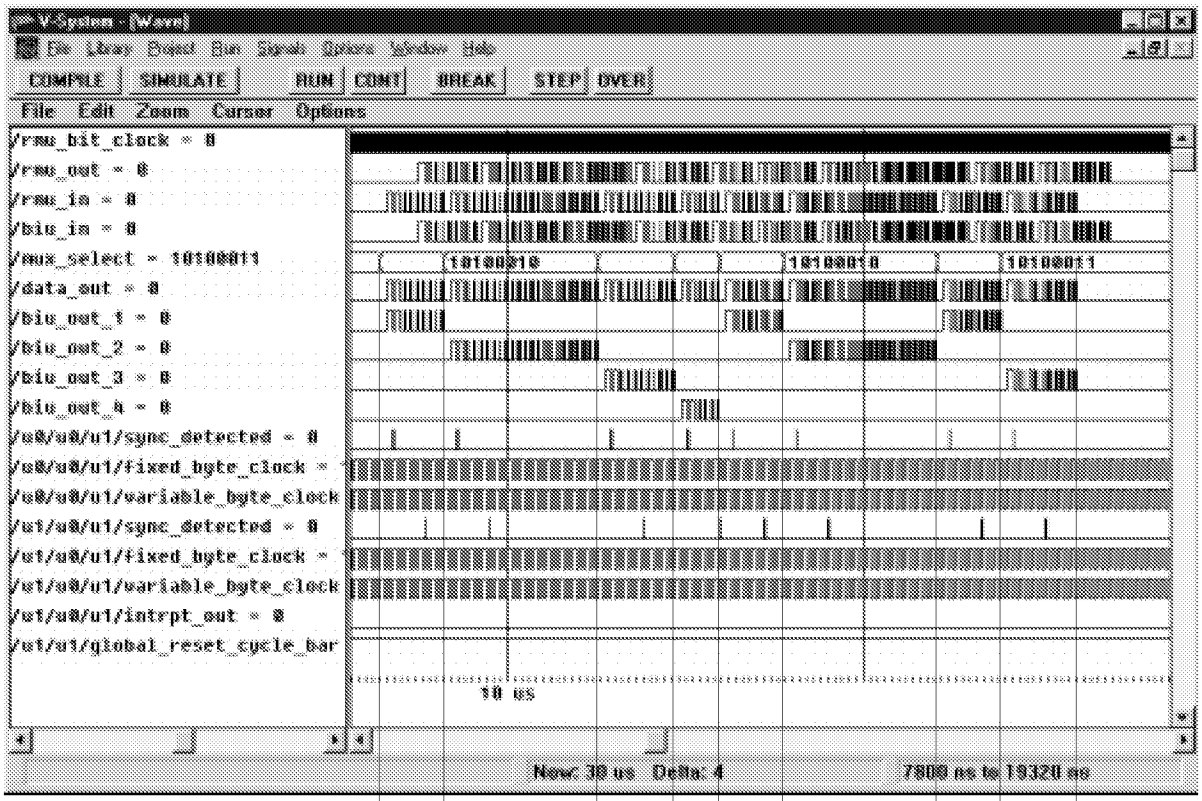


Figure 15. Typical scheduled activities.

In Figure 15 and subsequent figures, *rmu_out* is the output of the RMU that appears on the *read bus*. *rmu_in* is the input data to the RMU after multiplexing the BIU outputs from the *write buses*. *biu_in* is the same as *rmu_out* but at the input of the BIUs. *mux_select* indicates the selection value and hence the particular BIU output to be routed via the RMU. *biu_out_i* corresponds to the output of the BIU i that appear on its *write bus*. The horizontal axis is the time axis.

4.1 Ideal Case

In this case as shown in Figure 16, the system operation is shown under ideal conditions where the delays in the *read* and *write* busses are assumed to be zero and no fault exists. In this case, the BIU and RMU clocks are shown to be in perfect synchrony. The Schedule for this test case is listed in Appendix D.



Transmission slots for BIUs in one schedule cycle:	1	2	3	4	1	2	1	3

Figure 16. Ideal conditions.

Figure 17 and 18 are the details of Figure 16. In Figure 17 transmission of the BIU_3 can be traced to appear at the output of the RMU after a few clock ticks. In Figure 18 the Sync_Header appears at the output of RMU and is detected by the BIU_1. Upon resynchronization, BIU_1 issues an interrupt to its associated processing element.

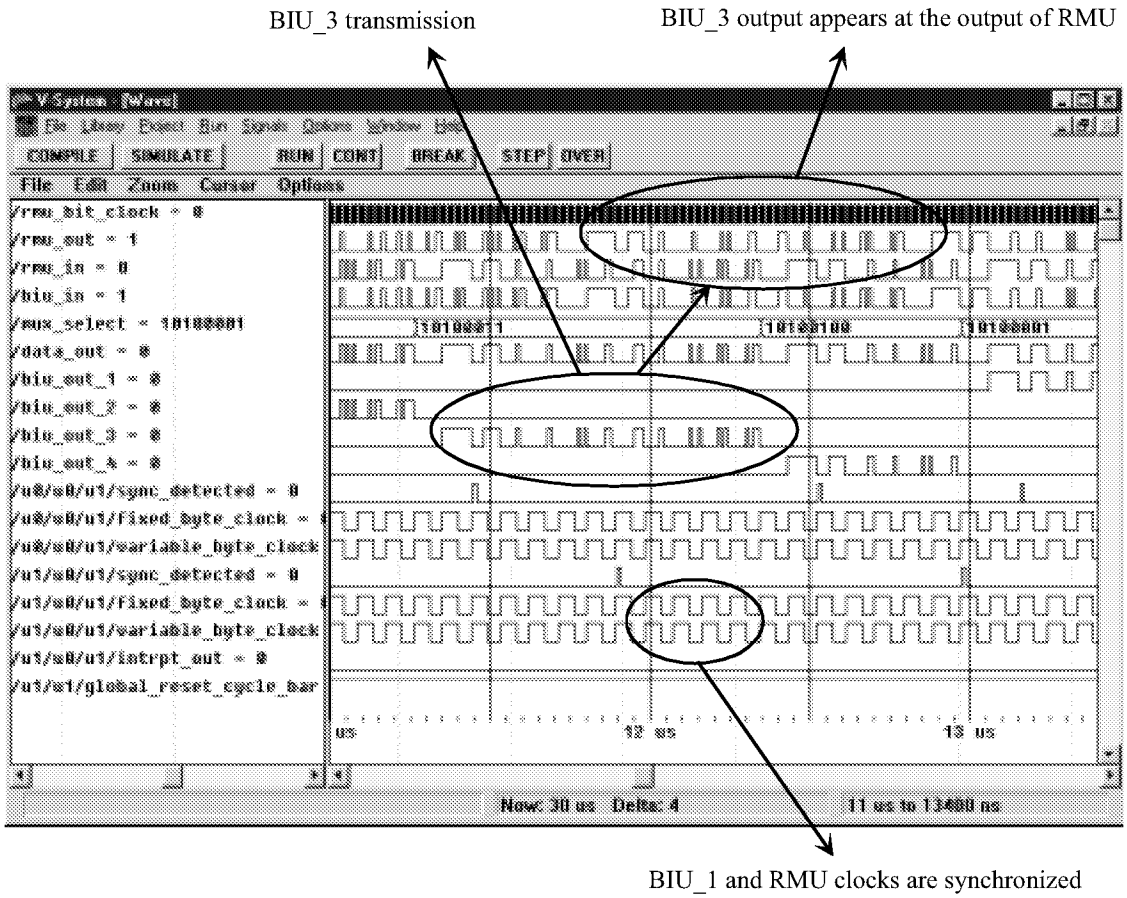


Figure 17. Ideal conditions, BIUs and RMU are in perfect synchrony.

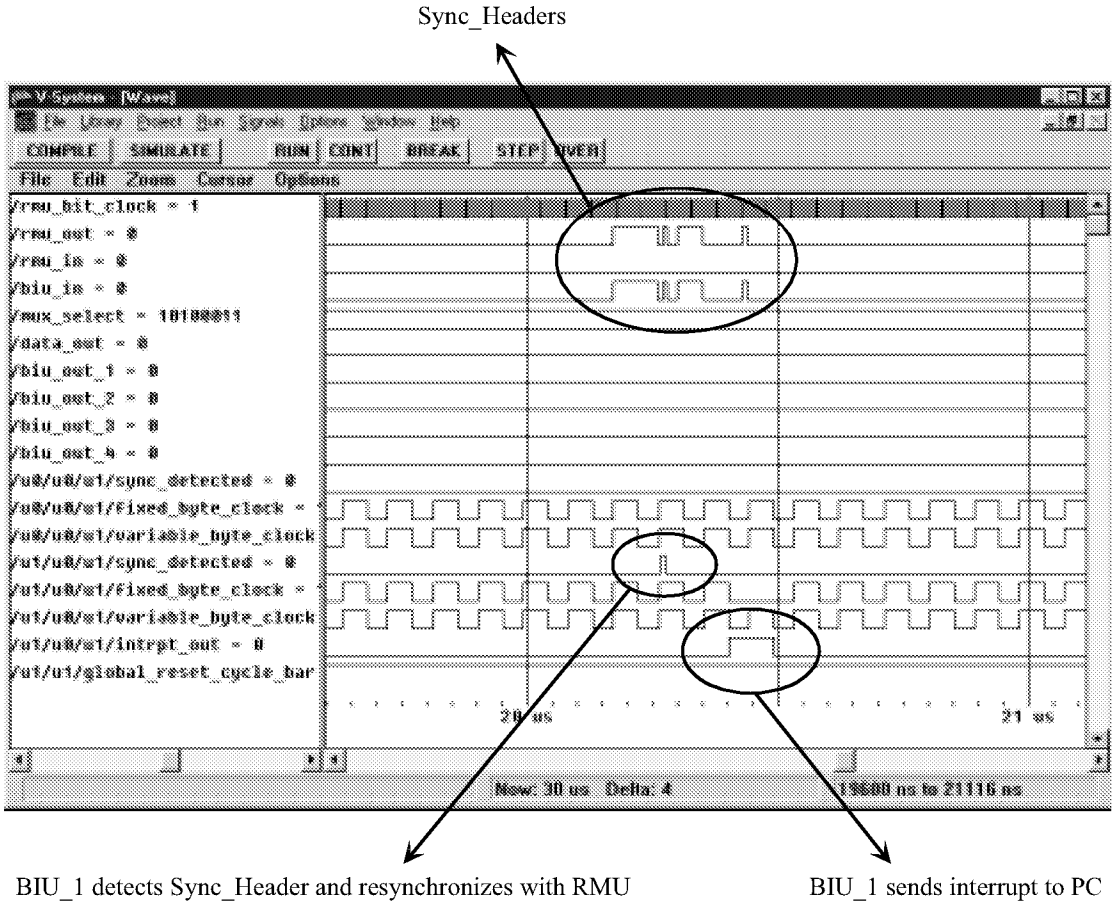


Figure 18. Start-Cycle command, BIU clocks are re-synchronized with the RMU clock.

4.2 Failing a BIU

Forcing the BIU to reset simulates, for example, failure of a BIU due to loss of power. In this test case, the system starts with all BIUs functioning normally. BIU 1 is then forced to reset in the middle of a scheduled period. As a result, BIU 1 (*biu_out_1* signal in Figure 19) stops executing scheduled instructions and is taken off line. Figure 19 depicts system activities for three consecutive scheduled cycles. As evident from Figure 19, BIU 1 (*biu_out_1*) stops transmitting for the rest of the second scheduled cycle. Figure 20 is a detailed picture of Figure 19 and depicts system activities for the duration of the second cycle. In case of loss of power, BIU 1 will remain off line. However, after it is powered on, BIU 1 will recover at the start of the next scheduled period. The BIUs have similar behavior in the case of reset. In other words, if a BIU is reset during normal operation, it will recover and join the system at the start of the next scheduled period although the RMU may choose to mask it out. Figure 21 provides the details for the recovery of BIU 1. The Schedule for this test case is listed in Appendix D.

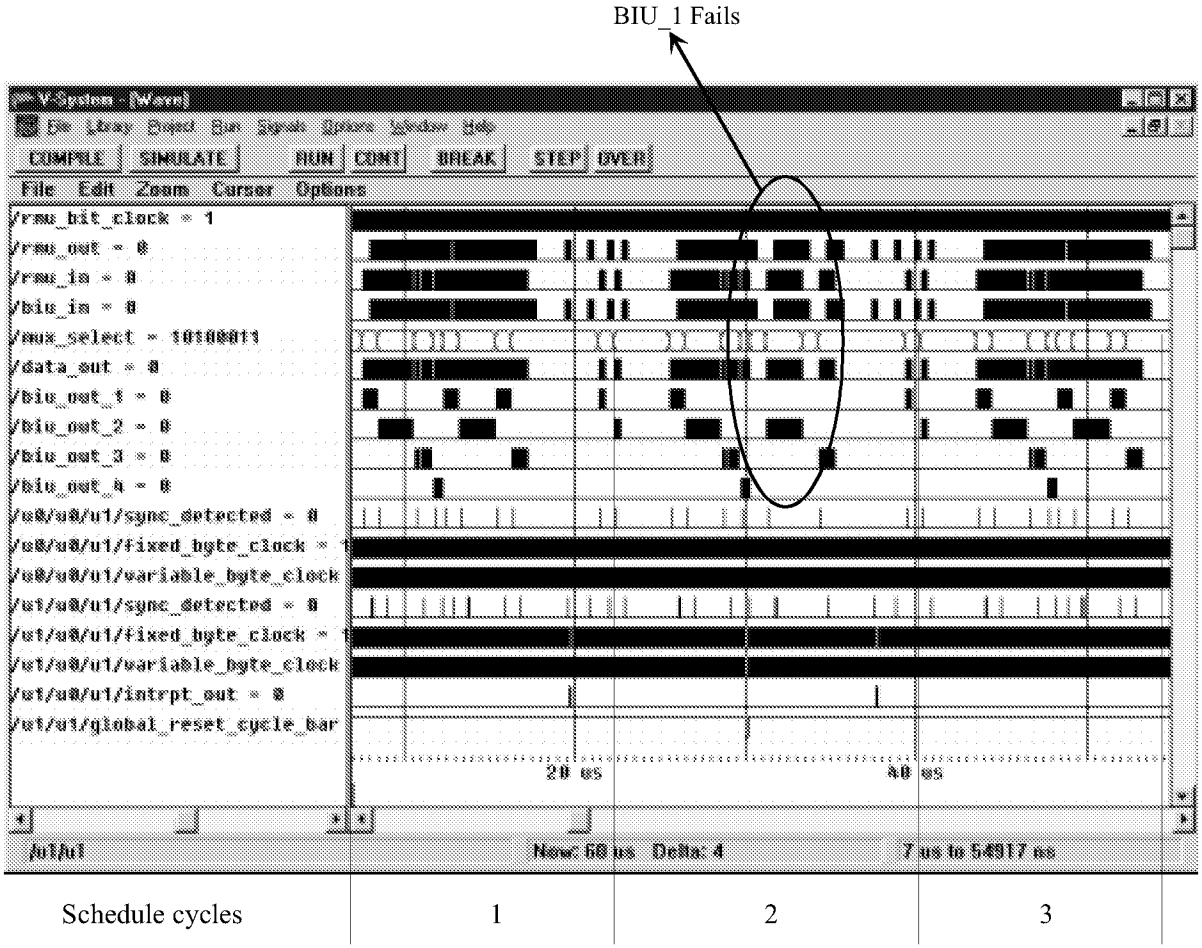


Figure 19. BIU 1 is powered down for one cycles.

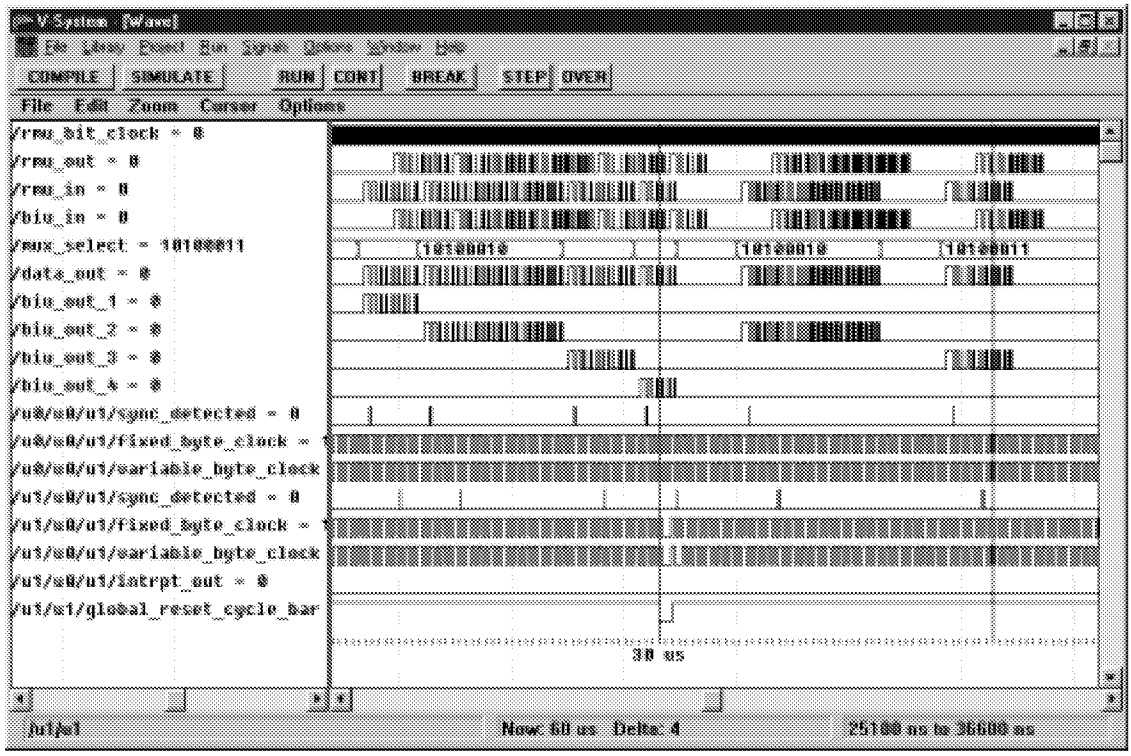


Figure 20. BIU 1 is powered down (detail).

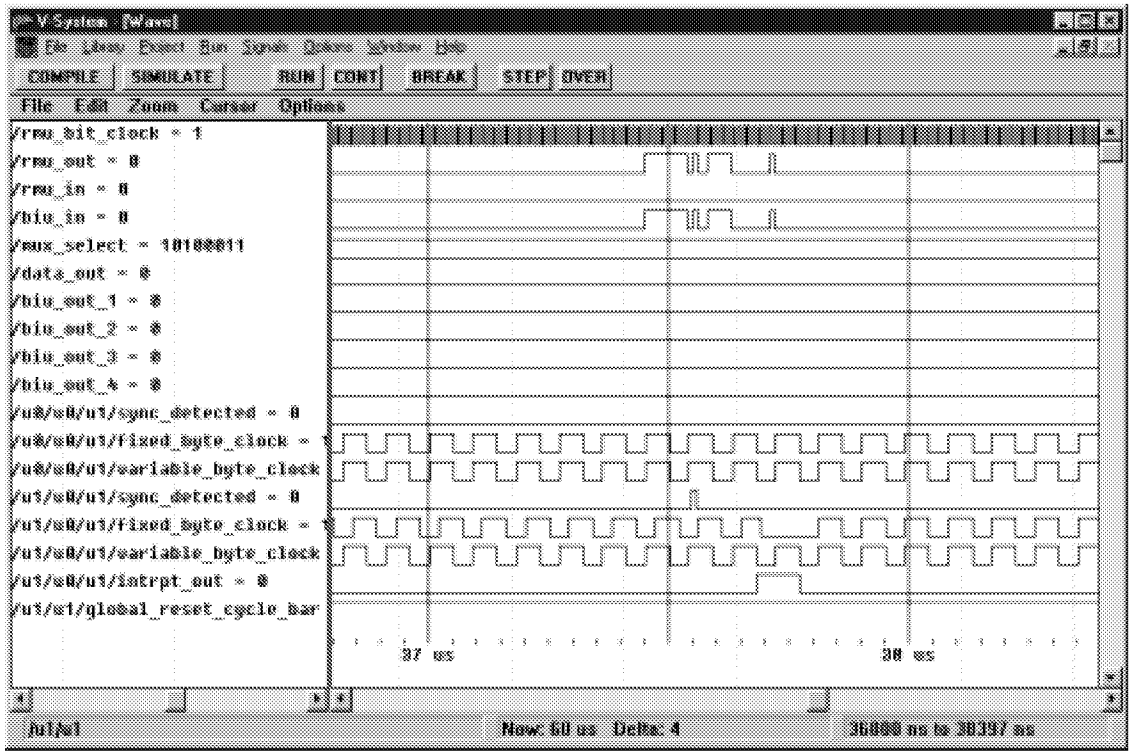


Figure 21. BIU 1 recovers, Start-Cycle command.

5. Summary

A single channel, fault-tolerant, fiber-optic backplane was developed to study the feasibility of the proposed architecture by Palumbo [1]. This backplane also assists with the investigations of behavior of the architecture in the presence of faults. The particular implementation of the architecture that is presented here enables a RMU to connect to as many as 29 BIUs; however, for testing purposes a maximum of four BIUs are sufficient to demonstrate full channel functionally. The architecture is designed, developed, and implemented using VHDL. Time constraints did not allow for a full hardware implementation; however, a large segment of the developed architecture is synthesized and implemented in hardware using Xilinx FPGAs on multiple prototype boards. The prototype boards are designed so that they can be configured to function as either a BIU or a RMU. Analysis of the test cases shows the feasibility of the backplane as well as backplane integrity in the presence of faults and recovery from faults.

5.1 Future Enhancements

There are two areas that require enhancements. The first is the design and development of a voter module for the RMU so that the backplane can be replicated and the proposed architecture can be studied in its entirety. The other enhancement is the introduction of a new parameter in the schedule, probably a third byte, to account for the variable length buses and to make the switch-time overhead minimal. This feature could replace the data arrival window currently implemented and thus maximize bus utilization. This parameter, delta time, needs to be associated with BIUs and its value needs to be an indication of the distance to the RMU so that the RMU switches the multiplexer after this delta time.

References

[1] Dan Palumbo: Fault-Tolerant Processing System. U.S. Patent Number 5,533,188, July 2, 1996.

[2] IEEE Standard VHDL Language Reference Manual, IEEE 1076-1987.

[3] Xilinx, "The Programmable Gate Array Data Books," 1992 and 1994.

Appendix A
VHDL Codes

File Conventions:

All modules are separated into entity and architecture pairs and are stored in separate files. The file name convention used is as follows:

filename_filetype.vhd

where, in order to maintain the file names compatibility on the PC and the workstation the *filename* is restricted to only six characters. The *filetype* is a single character and can be *e* for entities, *a* for architectures, *t* for testbenches, or *p* for packages. All files have the same *vhd* extensions. For instance, the FIFO module is stored in *fifo_e.vhd* and *fifo_a.vhd* files.

All files have a document section where the file attributes including the author, file name, file use, and all of the activities are chronologically described.

Naming Conventions:

The reserved words are in lower cases while the user defined names are either all in upper cases or at least the first character is in upper case. All user defined names are as descriptive as possible and underline characters are used to make them legible.

The I/O signals have one of the following forms:

Signal_Name_In for input signals,
Signal_Name_Out for output signals, and
Signal_Name_In_Out for input and output signals.

Active low signals are defined as:

Signal_Name_Bar

```
--------------------------------------------------------------------------------
-- File Name:       CNSTNT_2.VHD
-- Host Machine:    GATEWAY 486/33 (IBM AT Clone)
-- Target Machine:  GATEWAY 486/33 (IBM AT Clone)
-- Environment :    Model Technology VHDL Simulation for Windows (Ver 4.2e)
--                  DOS Version 6.2
-- Organization:    NASA-LaRC
-- Project:         Fly By Light - Power By Wire (FBL-PBW)
-- Author:          Mahyar R. Malekpour
-- Creation Date:   3/19/96
--------------------------------------------------------------------------------
-- Name/Number:
--    CNSTNT_P.VHD                                  (entity/architecture)
--
-- Abstract:
--
-- Acronyms/Abbreviations:
--   FBL/PBW
--
-- Dependencies:
--   none
--
-- Global Objects:
--
-- Exceptions:
--
-- Machine/Compiler Dependencies:
--
-- Revisions:
--   Modified on:  4/9/1996
--   by:           Mahyar Malekpour
--
-- Address            Device         Use
-- ..........................................................
-- PORT_ADDRESS_0     XC3020         Read/Write FIFOs
-- PORT_ADDRESS_1     XC3020         Read Status of FIFO
-- PORT_ADDRESS_2     XC3020         Write Status (reset FIFOs)
-- PORT_ADDRESS_3     XC3020          Transfer data between FIFOs
-- PORT_ADDRESS_4     Not used
-- PORT_ADDRESS_5     XC3020         Reset and Program XC4000
-- PORT_ADDRESS_6     PAL            Reset XC3020
-- PORT_ADDRESS_7     PAL            Program XC3020
--
--
-- Revisions:
--   Modified on:  4/18/1996
--   by:           Mahyar Malekpour
-- Changed port names here instead of in the INTRFC_A.VHD file.
--
--   Modified on:  8/9/1996
--   by:           Mahyar Malekpour
-- Added Data_Length_Plus_1 and Sync_Pattern.
--
--------------------------------------------------------------------------------
library IEEE ;
use IEEE.std_logic_1164.all ;

package CNSTNT_P is

    ------------------------------------------------------------------------
    constant     PORT_Length : integer := 2 ;

    constant       Data_ADDRESS : std_logic_vector (PORT_Length downto 0) := "000" ;
    constant     Status_ADDRESS : std_logic_vector (PORT_Length downto 0) := "001" ;
    constant    Command_ADDRESS : std_logic_vector (PORT_Length downto 0) := "010" ;
    constant     PORT_ADDRESS_3 : std_logic_vector (PORT_Length downto 0) := "011" ;
    constant     PORT_ADDRESS_4 : std_logic_vector (PORT_Length downto 0) := "100" ;
    constant  PROG_4000_ADDRESS : std_logic_vector (PORT_Length downto 0) := "101" ;
    constant     PORT_ADDRESS_6 : std_logic_vector (PORT_Length downto 0) := "110" ;
    constant     PORT_ADDRESS_7 : std_logic_vector (PORT_Length downto 0) := "111" ;

    ------------------------------------------------------------------------
    constant    BASE_ADDRESS : std_logic_vector (6 downto 0) := "1100000" ; -- 300 thru 307

    constant           Data_Length : integer := 7 ;
    constant    Data_Length_Plus_1 : integer := Data_Length + 1 ;
```

```
    constant  Transmit_Byte_Length : integer := 9 ;

    constant       Sync_Pattern : std_logic_vector (Data_Length_Plus_1 downto 0) := "111111110" ;

    ------------------------------------------------------------------------
    -- BIU_ID_0 is reserved and should not be used.
    -- Only the lower 5 bits are part of the ID and higher 3 bits are reserved.
    -- Thus, there are a total of 32 - 2 = 30 BIU/RMUs in a channel.
    -- Minus 2 because ID = 0 is ignored and ID = 31 is a global id.
    -- 8/20/96
    constant  BIU_ID_0 : std_logic_vector (Data_Length downto 0) := "00000000" ;
    constant  BIU_ID_1 : std_logic_vector (Data_Length downto 0) := "00000001" ;
    constant  BIU_ID_2 : std_logic_vector (Data_Length downto 0) := "00000010" ;
    constant  BIU_ID_3 : std_logic_vector (Data_Length downto 0) := "00000011" ;
    constant  BIU_ID_4 : std_logic_vector (Data_Length downto 0) := "00000100" ;
    constant  BIU_ID_5 : std_logic_vector (Data_Length downto 0) := "00000101" ;
    constant  BIU_ID_6 : std_logic_vector (Data_Length downto 0) := "00000110" ;
    constant  BIU_ID_7 : std_logic_vector (Data_Length downto 0) := "00000111" ;
    constant  BIU_ID_8 : std_logic_vector (Data_Length downto 0) := "00001000" ;

    constant  RMU_ID_1 : std_logic_vector (Data_Length downto 0) := "00011011" ; -- 27
    constant  RMU_ID_2 : std_logic_vector (Data_Length downto 0) := "00011100" ; -- 28
    constant  RMU_ID_3 : std_logic_vector (Data_Length downto 0) := "00011101" ; -- 29
    constant  RMU_ID_4 : std_logic_vector (Data_Length downto 0) := "00011110" ; -- 30

    constant  Global_BIU_ID : std_logic_vector (Data_Length downto 0) := "00011111" ; -- 31

    ------------------------------------------------------------------------
    constant Bit_Clock_Period : time := 10 ns ;

    constant    Delay_2_ns : time := 2 ns ;
    constant    Delay_5_ns : time := 5 ns ;
    constant    Delay_7_ns : time := 7 ns ;
    constant   Delay_10_ns : time := 10 ns ;
    constant   Delay_12_ns : time := 12 ns ;
    constant   Delay_15_ns : time := 15 ns ;
    constant   Delay_20_ns : time := 20 ns ;
    constant   Delay_25_ns : time := 25 ns ;
    constant   Delay_30_ns : time := 30 ns ;

    ------------------------------------------------------------------------

end CNSTNT_P ;

--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
-- File Name:       PAL22V_E.VHD
-- Host Machine:    GATEWAY 486/33 (IBM AT Clone)
-- Target Machine:  GATEWAY 486/33 (IBM AT Clone)
-- Environment :    Model Technology VHDL Simulation for Windows (Ver 4.3f)
--                  DOS Version 6.2
-- Organization:    NASA-LaRC
-- Project:         Fly By Light - Power By Wire (FBL-PBW)
-- Author:          Mahyar R. Malekpour
-- Creation Date:   09/21/95
--------------------------------------------------------------------------------
-- Name/Number:
--    PAL22V                                (entity)
--
-- Abstract:
--   This file contains the entity declaration for the PC interface that will
-- be programmed on a PALL22V10.
--
--   SIGNAL DEFINITION :
--
--
-- Acronyms/Abbreviations:
--   FBL/PBW
--   BIU - Bus Interface Unit
--
-- Dependencies:
--   IEEE.STD_LOGIC_1164
--
-- Global Objects:
--
```

```
-- Exceptions:
--
-- Machine/Compiler Dependencies:
--
-- Revisions:
--
--   Modified on:  10/12/95
--   by:           Mahyar Malekpour
--
--   1.  Added CLK_In signal to this entity for use by the D flip-flops.
--
--   2.  Added X_CLK_Out signal to separate the reset-port and program-port
-- operations.  The CLK_In is yied to the reset-port and thus to CLK_Out,
-- while the X_CLK_out is tied to program-port and is generated for the
-- Xilinx chip.
--
--
--   Modified on:  10/16/95
--   by:           Mahyar Malekpour
--
--   1.  "CLK_In" must be hooked up to pin "1" of the PAL22V10, it is the
-- clock pin of all flip-flops inside the PAL.
--
--   2.  "CLK_Out" must be hooked up to "CLK_In".  It is the feedback clock
-- generated by the internal logic of the PAL and is used to latch in D0 and
-- D1 signals.
--
--   3.  "Done_Prog_Bar" must be hooked up to "FeedBack_Done_Prog".  It is
-- the feedback for tri-stating the input signal.  The Picdesign was wasting
-- too much of the internal logic blocks and I/O pins beyound our
-- expectations and was requiring another PAL to do the job!  By manually
-- feeding this signal back to the PAL I managged to tri-state it without
-- additional PAL and saved a lot of I/O pins in the current PAL.
--
--
--------------------------------------------------------------------------------

library IEEE;
use IEEE.std_logic_1164.all;

entity PAL22V is
    port (
           -- Inputs
           CLK_In             : in  std_logic ;
           ADDRESS            : in  std_logic_vector (9 downto 0) := (others => '0') ;
           AEN                : in  std_logic ; -- Address enable, active high
           IOWR_Bar           : in  std_logic ;
           IORD_Bar           : in  std_logic ;
           RESET              : in  std_logic ; -- Power on reset, active high
           D0                 : in  std_logic ;
           D1                 : in  std_logic ;
           INIT               : in  std_logic ;
           FeedBack_Done_Prog : in  std_logic ;

           --  Outputs
           Done_Prog_Tristate : out std_logic ;
           CLK_Out            : out std_logic ;
           X_CLK_Out          : out std_logic ;
           Data_Out           : out std_logic ;
           Reset_Out          : out std_logic ;

           --  In/Outputs
           Done_Prog_Bar      : out std_logic ) ;

end PAL22V ;

--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
-- File Name:       PAL22V_A.VHD
-- Host Machine:    GATEWAY 486/33 (IBM AT Clone)
-- Target Machine:  GATEWAY 486/33 (IBM AT Clone)
-- Environment :    Model Technology VHDL Simulation for Windows (Ver 4.3f)
--                  DOS Version 6.2
-- Organization:    NASA-LaRC
-- Project:         Fly By Light - Power By Wire (FBL-PBW)
```

```
-- Author:          Mahyar R. Malekpour
-- Creation Date:   09/21/95
--------------------------------------------------------------------------------
-- Name/Number:
--    PAL22V                                  (architecture)
--
-- Abstract:
--   This file contains the architecture for the PC interface that will
-- be programmed on a PALL22V10.
--
--   SIGNAL DEFINITION :
--
--
-- Acronyms/Abbreviations:
--   FBL/PBW
--   BIU - Bus Interface Unit
--
-- Dependencies:
--   IEEE.STD_LOGIC_1164
--
-- Global Objects:
--
-- Exceptions:
--
-- Machine/Compiler Dependencies:
--
-- Revisions:
--
--   Modified on:  10/3/95
--   by:           Mahyar Malekpour
--
--   1.  The Xilinx program-port is at address 301 Hex.
--
--   2.  The reset-port is at address 300 Hex.
--
--   3.  Renamed the Reset_Out_Bar signal to Reset_Out for it is a user
-- programmable signal.  Reset_Out signal is tied to the power on RESET
-- signal and the D1 signal.  Through D1, it can be programmed to stay high
-- or low provided the reset-port is addressed.  The reset-port address is
-- 300 Hex, for now.
--
--   4.  Built a latch for the D1 signal so that the Reset_Out signal can be
-- user programmable.
--
--   Modified on:  10/12/95
--   by:           Mahyar Malekpour
--
--   1.  The latch is not implementable on the PAL via the Cadence PicDesign
-- tools.  Since the PAL has D flip-flops, I have redefined the latch
-- construct as a D flip-flop for the D1 signal.
--
--   2.  Added CLK_In signal to the entity of this architecture.
--   By feeding back the CLK_Out signal generated by the PAL to the PAL via
-- the CLK_In signal (pin 1), the CLK_In signal could be used to clock
-- (latch) the D1 signal.  A good test of the tools used (Cadence PicDesign
-- here) is that it should tie the CLK_In signal to pin 1 of the PAL.
--
--   Note: The CLK_In signal assignment to pin 1 should never be altered.
--
--   3.  Added X_CLK_Out signal to separate the reset-port and program-port
-- operations.  The CLK_In is tied to the reset-port and thus to CLK_Out,
-- while the X_CLK_out is tied to program-port and is generated for the
-- Xilinx chip.
--
--   Modified on:  3/12/96
--   by:           Mahyar Malekpour
--
--   1.  The Xilinx program-port is at NEW address 307 Hex.
--
--   2.  The reset-port is at NEW address 306 Hex.
--
--------------------------------------------------------------------------------

library IEEE ;
use IEEE.std_logic_1164.all ;
```

```
architecture PAL22V_Behaviour of PAL22V is

    signal  PORT_1_SELECTED : std_logic ;
    signal  PORT_2_SELECTED : std_logic ;
constant PORT_1_ADDRESS : std_logic_vector (9 downto 0) := "1100000110" ;
    constant PORT_2_ADDRESS : std_logic_vector (9 downto 0) := "1100000111" ;

begin

    ------------------------------------------------------------------------
    Check_Addresses : process ( ADDRESS, AEN, IOWR_Bar, IORD_Bar )
      variable TEMP, TEMP2, TEMP3 : std_logic := '0' ;
    begin
      TEMP2 := (not AEN) and (not IOWR_Bar) ;

      if ( ADDRESS = PORT_1_ADDRESS ) then
        TEMP := '1' ;
      else
        TEMP := '0' ;
      end if ;

      -- writing to reset-port
      PORT_1_SELECTED <= TEMP and TEMP2 ;

      if ( ADDRESS = PORT_2_ADDRESS ) then
        TEMP := '1' ;
      else
        TEMP := '0' ;
      end if ;

      -- writing to program-port
      PORT_2_SELECTED <= TEMP and TEMP2 ;

    end process ;

    ------------------------------------------------------------------------
    -- Need to be able to reset the Xilinx for longer than one write cycle.
    -- Therefore, we need to latch the D1 signal that is used to reset the
    -- Xilinx.
    --
    Latch_Process : process ( CLK_In )
    begin
      if ( Rising_Edge ( CLK_In )) then
        Reset_Out <= D0 ;
      end if ;
    end process ;

    ------------------------------------------------------------------------

    CLK_Out <= not ( PORT_1_SELECTED ) ;

    X_CLK_Out <= not ( PORT_2_SELECTED ) ; -- not (PORT_SELECTED and (not IOWR_Bar)) ;

    Data_Out <= D0 ;

    Latch_DP_process : process ( CLK_In )
    begin
      if ( Rising_Edge ( CLK_In )) then
        Done_Prog_Bar <= D1 ;
      end if ;
    end process ;

    ------------------------------------------------------------------------

    Tri_State_Process : process ( FeedBack_Done_Prog )
    begin
      if ( FeedBack_Done_Prog = '1' ) then
        Done_Prog_Tristate <= '0' ;
      else
        Done_Prog_Tristate <= 'Z' ;
      end if ;

    end process ;
    ------------------------------------------------------------------------
```

```
end PAL22V_Behaviour ;

--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
-- File Name:       FIFO_E.VHD
-- Host Machine:    GATEWAY 486/33 (IBM AT Clone)
-- Target Machine:  GATEWAY 486/33 (IBM AT Clone)
-- Environment :    Model Technology VHDL Simulation for Windows (Ver 4.3f)
--                  DOS Version 6.2
-- Organization:    NASA-LaRC
-- Project:         Fly By Light - Power By Wire (FBL-PBW)
-- Author:          Mahyar R. Malekpour
-- Creation Date:   7/22/1996
--------------------------------------------------------------------------------
-- Name/Number:
--    FIFO                                (entity)
--
-- Abstract:
--   This file contains the entity declaration for a generic FIFO.
-- It conforms with the FIFO chip used in our board, i.e., AM???
--
--   SIGNAL DEFINITION :
--
-- Full_Flag_Bar  -- active low,  '1' ==> not full, '0' ==> full
-- Empty_Flag     -- active high, '0' ==> empty, '1' ==> not empty
-- HF_Flag_Bar    -- active low,  '1' ==> not half full, '0' ==> half full
--
-- Acronyms/Abbreviations:
--
-- Dependencies:
--   IEEE.STD_LOGIC_1164
--
-- Global Objects:
--
-- Exceptions:
--
-- Machine/Compiler Dependencies:
--
-- Revisions:
--
--   Modified on:  ??/??/96
--   by:           Mahyar Malekpour
--
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------

library IEEE ;
use IEEE.std_logic_1164.all ;

entity FIFO is
    generic (
           Period         : time := 100 ns ;
           Depth          : natural := 10 ; -- 2 K for now
           Width          : natural := 7 ) ; -- 8-bit Byte

    port (
           Data_In        : in    std_logic_vector (Width downto 0) ;
           Data_Out       : out   std_logic_vector (Width downto 0) := "ZZZZZZZZ" ;

           Reset_Bar      : in    std_logic ; -- := '1' ;
           Read_Bar       : in    std_logic ; -- := '1' ;
           Write_Bar      : in    std_logic ; -- := '1' ;

           Full_Flag_Bar  : out   std_logic ; -- active low
           Empty_Flag     : out   std_logic ; -- active high
           HF_Flag_Bar    : out   std_logic   -- active low

         ) ;

end FIFO ;

--------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------
-------------------------------------------------------
-- File Name:       FIFO_A.VHD
```

```
-- Host Machine:    GATEWAY 486/33 (IBM AT Clone)
-- Target Machine:  GATEWAY 486/33 (IBM AT Clone)
-- Environment :    Model Technology VHDL Simulation for Windows (Ver 4.3f)
--                  DOS Version 6.2
-- Organization:    NASA-LaRC
-- Project:         Fly By Light - Power By Wire (FBL-PBW)
-- Author:          Mahyar R. Malekpour
-- Creation Date:   7/22/1996
--------------------------------------------------------------------------------
-- Name/Number:
--    FIFO                                (entity)
--
-- Abstract:
--   This file contains the entity declaration for a generic FIFO.
-- It conforms with the FIFO chip used in our board, i.e., AM???
--
--   SIGNAL DEFINITION :
--
-- Full_Flag_Bar  -- active low,  '1' ==> not full, '0' ==> full
-- Empty_Flag     -- active high, '0' ==> empty, '1' ==> not empty
-- HF_Flag_Bar    -- active low,  '1' ==> not half full, '0' ==> half full
--
-- Acronyms/Abbreviations:
--
-- Dependencies:
--   IEEE.STD_LOGIC_1164
--
-- Global Objects:
--
-- Exceptions:
--
-- Machine/Compiler Dependencies:
--
-- Revisions:
--
--   Modified on:  8/9/96
--   by:           Mahyar Malekpour
-- Fine tuned a bit more today.
--
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------

library IEEE ;
use IEEE.std_logic_1164.all ;
use ieee.std_logic_arith.all ;
use work.my_std_logic_arith.all ;

architecture FIFO_Behave of FIFO is

    ------------------------------------------------------------------------
    type Memory is array (0 to Depth - 1) of integer ;
    signal FIFO_Memory : Memory ;

begin

    ------------------------------------------------------------------------
    process ( Read_Bar, Write_Bar, Reset_Bar )

      variable Delay     : time := Period / 3 ;
      variable Count     : natural range 0 to Depth := 0 ;
      variable Read_Ptr  : natural range 0 to Depth := 0 ;
      variable Write_Ptr : natural range 0 to Depth := 0 ;

      variable TEMP      : integer := 0 ;

    begin

      if (Read_Bar = '1') and (Write_Bar = '1') and (Reset_Bar = '0') then
        Write_Ptr := 0 ;
        Read_Ptr  := 0 ;
        Count     := 0 ;
        Full_Flag_Bar <= '1' after Delay ;
        Empty_Flag    <= '0' after Delay ;
        HF_Flag_Bar   <= '1' after Delay ;
        Data_Out <= "ZZZZZZZZ" ; -- a must here
```

```
      elsif ( Reset_Bar = '1' ) then
        if ( Falling_Edge ( Write_Bar ) and ( Count < Depth )) then
          Count := Count + 1 ;
          TEMP := To_Integer ( Data_In ) ;
          FIFO_Memory ( Write_Ptr ) <= TEMP ;

          Write_Ptr := ( Write_Ptr + 1 ) mod Depth ;
        end if ;

        if ( Falling_Edge ( Read_Bar ) and ( Count > 0 )) then
          TEMP := FIFO_Memory ( Read_Ptr ) ;
          Data_Out <= To_StdLogicVector ( TEMP, 8 ) after 10 ns ;

          Count := Count - 1 ;
          Read_Ptr := ( Read_Ptr + 1 ) mod Depth ;

        elsif Rising_Edge ( Read_Bar ) then
          Data_Out <= "ZZZZZZZZ" after 10 ns ; -- a must

        end if ;

        if ( Count = 0 ) then
          Empty_Flag <= '0' after Delay ;
        else
          Empty_Flag <= '1' after Delay ;
        end if ;

        if ( Count >= Depth ) then
          Full_Flag_Bar <= '0' after Delay ;
        else
          Full_Flag_Bar <= '1' after Delay ;
        end if ;

        if ( Count >= Depth / 2 ) then
          HF_Flag_Bar <= '0' after Delay ;
        else
          HF_Flag_Bar <= '1' after Delay ;
        end if ;

    end if ;

    end process ;

    -------------------------------------------------------------------------

end FIFO_Behave ;
-------------------------------------------------------------------------------
-------------------------------------------------------------------------------
-------------------------------------------------------------------------------
-- File Name:       XC3020_E.VHD
-- Host Machine:    GATEWAY 486/33 (IBM AT Clone)
-- Target Machine:  GATEWAY 486/33 (IBM AT Clone)
-- Environment :    Model Technology VHDL Simulation for Windows (Ver 4.3f)
--                  DOS Version 6.2
-- Organization:    NASA-LaRC
-- Project:         Fly By Light - Power By Wire (FBL-PBW)
-- Author:          Mahyar R. Malekpour
-- Creation Date:   3/19/96
-------------------------------------------------------------------------------
-- Name/Number:
--    XC3020                                (entity)
--
-- Abstract:
--   This file contains the entity declaration for the PC interface and part
-- of the BIU that will be programmed on a Xilinx XC3020.
--
--   SIGNAL DEFINITION :
--
--
-- Acronyms/Abbreviations:
--   FBL/PBW
--   BIU - Bus Interface Unit
--
-- Dependencies:
--   IEEE.STD_LOGIC_1164
--
```

```
-- Global Objects:
--
-- Exceptions:
--
-- Machine/Compiler Dependencies:
--
-- Revisions:
--
--   Modified on:  3/25/1996
--   by:           Mahyar Malekpour
--   Added Chip_Select_Bar signal to the entity.  It is needed in the XC4000.
--
--   Modified on:  4/9/1996
--   by:           Mahyar Malekpour
--   Added CCLK_4000, Din_4000, Prog_4000, INIT_4000, and DONE_4000 signals
-- to the entity.  They are used for programming of the XC4000 in both the
-- slave serial mode and parallel periferal mode.
--
--   Modified on:  6/7/1996
--   by:           Mahyar Malekpour
-- Added Direction signal to the entity to control flow of data thru 74LS245
-- bidirectional buffer that connects uP_Data bus to the XC3020.  The dehault
-- value of Direction signal is high, i.e., uP is writing, otherwise low only
-- when uP is reading from ports within the xc3020.
--
--------------------------------------------------------------------------------

library IEEE ;
use IEEE.std_logic_1164.all ;
use WORK.CNSTNT_P.all ;

entity XC3020 is
    port (
-- the following 4 signals are not synthesizable and so are commented out.
-- 3/19/96
--           RESET                : in    std_logic ; -- Power on reset, active high
--           CLK_In               : in    std_logic ;
--           Serial_Prog_In       : in    std_logic ;
--           Done_Prog_Bar        : in    std_logic ;

           Reset_BIU            : out   std_logic ;
           Direction            : out   std_logic ;
           Data_Read_Bar        : out   std_logic ;
           Data_Write_Bar       : out   std_logic ;

           -- This signal is added because it is needed in the XC4000
           -- Mahyar 3/25/1996
           Chip_Select_Bar      : out   std_logic ;

           ADDRESS              : in    std_logic_vector (9 downto 0) ;
           AEN_Bar              : in    std_logic ; -- Address enable, active high
           IOWR_Bar             : in    std_logic ;
           IORD_Bar             : in    std_logic ;

           uP_Data_In_Out       : inout std_logic_vector (Data_Length downto 0) ;
           FIFO_Data_In_Out     : inout std_logic_vector (Data_Length downto 0) ;

           Input_FIFO_Reset_Bar  : out   std_logic ;
           Input_FIFO_Read_Bar   : out   std_logic ;
           Input_FIFO_Write_Bar  : out   std_logic ;
           Input_FIFO_Full_Bar   : in    std_logic ;
           Input_FIFO_Empty_Bar  : in    std_logic ;
           Input_FIFO_HF_Bar     : in    std_logic ;

           Output_FIFO_Reset_Bar : out   std_logic ;
           Output_FIFO_Read_Bar  : out   std_logic ;
           Output_FIFO_Write_Bar : out   std_logic ;
           Output_FIFO_Full_Bar  : in    std_logic ;
           Output_FIFO_Empty_Bar : in    std_logic ;
           Output_FIFO_HF_Bar    : in    std_logic ;

           BIU_FIFO_Read_Bar     : in    std_logic ;
           BIU_FIFO_Write_Bar    : in    std_logic ;

           CCLK_4000             : out   std_logic ;
           Prog_4000             : out   std_logic ;
```

```
          INIT_4000             : in    std_logic ;
          DONE_4000             : in    std_logic

        ) ;

end XC3020 ;

--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
-- File Name:       XC3020_A.VHD
-- Host Machine:    GATEWAY 486/33 (IBM AT Clone)
-- Target Machine:  GATEWAY 486/33 (IBM AT Clone)
-- Environment :    Model Technology VHDL Simulation for Windows (Ver 4.3f)
--                  DOS Version 6.2
-- Organization:    NASA-LaRC
-- Project:         Fly By Light - Power By Wire (FBL-PBW)
-- Author:          Mahyar R. Malekpour
-- Creation Date:   3/19/96
--------------------------------------------------------------------------------
-- Name/Number:
--    XC3020                                (architecture)
--
-- Abstract:
--   This file contains the architecture for the PC interface and part
-- of the BIU that will be programmed on a Xilinx XC3020.
--
--   SIGNAL DEFINITION :
--
--
-- Acronyms/Abbreviations:
--   FBL/PBW
--   BIU - Bus Interface Unit
--
-- Dependencies:
--   IEEE.STD_LOGIC_1164
--
-- Global Objects:
--
-- Exceptions:
--
-- Machine/Compiler Dependencies:
--
-- Revisions:
--
--   Modified on:  4/9/1996
--   by:           Mahyar Malekpour
-- Modified the entities to reflect the newly added XC4000 related signals.
--
--   Modified on:  6/7/1996
--   by:           Mahyar Malekpour
-- Modified the entities to reflect the newly added Direction signal.
--
--------------------------------------------------------------------------------

library IEEE ;
use IEEE.std_logic_1164.all ;
use WORK.CNSTNT_P.all ;

architecture XC3020_Behave of XC3020 is

    --------------------------------------------------------------------------
    component INTRFC
    port (
          Reset_BIU             : out   std_logic ;
          Direction             : out   std_logic ;
          Data_Read_Bar         : out   std_logic ;
          Data_Write_Bar        : out   std_logic ;

          Chip_Select_Bar       : in    std_logic ; -- chip select, active low

          ADDRESS               : in    std_logic_vector (PORT_Length downto 0) ;
          IOWR_Bar              : in    std_logic ;
          IORD_Bar              : in    std_logic ;

          uP_Data_In_Out        : inout std_logic_vector (Data_Length downto 0) ;
```

```
          FIFO_Data_In_Out      : inout std_logic_vector (Data_Length downto 0) ;

          Input_FIFO_Reset_Bar  : out   std_logic ;
          Input_FIFO_Read_Bar   : out   std_logic ;
          Input_FIFO_Write_Bar  : out   std_logic ;
          Input_FIFO_Full_Bar   : in    std_logic ;
          Input_FIFO_Empty_Bar  : in    std_logic ;
          Input_FIFO_HF_Bar     : in    std_logic ;

          Output_FIFO_Reset_Bar : out   std_logic ;
          Output_FIFO_Read_Bar  : out   std_logic ;
          Output_FIFO_Write_Bar : out   std_logic ;
          Output_FIFO_Full_Bar  : in    std_logic ;
          Output_FIFO_Empty_Bar : in    std_logic ;
          Output_FIFO_HF_Bar    : in    std_logic ;

          BIU_FIFO_Read_Bar     : in    std_logic ;
          BIU_FIFO_Write_Bar    : in    std_logic ;

          CCLK_4000             : out   std_logic ;
          Prog_4000             : out   std_logic ;
          INIT_4000             : in    std_logic ;
          DONE_4000             : in    std_logic

        ) ;
    end component ;

    ------------------------------------------------------------------------
    component uP_PRT
    port (
          ADDRESS               : in    std_logic_vector (9 downto 0) ;
          AEN_Bar               : in    std_logic ; -- Address enable, active high
          Chip_Select_Bar       : out   std_logic   -- chip select, active low

        ) ;
    end component ;

    ------------------------------------------------------------------------
    for all : INTRFC use entity work.INTRFC (INTRFC_Behaviour) ;
    for all : uP_PRT use entity work.uP_PRT (uP_PRT_Behave) ;

    ------------------------------------------------------------------------
    -- for INTRFC

    -- for uP_PRT
    signal Chip_Select : std_logic ;

    ------------------------------------------------------------------------

begin

    U0 : INTRFC  port map ( Reset_BIU, Direction, Data_Read_Bar,
                            Data_Write_Bar, Chip_Select,
                            ADDRESS (PORT_Length downto 0),
                              IOWR_Bar, IORD_Bar, uP_Data_In_Out,
                                 FIFO_Data_In_Out,
                            Input_FIFO_Reset_Bar, Input_FIFO_Read_Bar,
                            Input_FIFO_Write_Bar,
                            Input_FIFO_Full_Bar, Input_FIFO_Empty_Bar,
                            Input_FIFO_HF_Bar,
                            Output_FIFO_Reset_Bar, Output_FIFO_Read_Bar,
                            Output_FIFO_Write_Bar,
                            Output_FIFO_Full_Bar, Output_FIFO_Empty_Bar,
                            Output_FIFO_HF_Bar,
                            BIU_FIFO_Read_Bar, BIU_FIFO_Write_Bar,
                            CCLK_4000, Prog_4000, INIT_4000, DONE_4000 ) ;

    U1 : uP_PRT  port map ( ADDRESS, AEN_Bar, Chip_Select ) ;

    -- Send it out to the XC4000 as well
    Chip_Select_Bar <= Chip_Select ;

end XC3020_Behave ;

------------------------------------------------------------------------
------------------------------------------------------------------------
```

```
--------------------------------------------------------------------------------
-- File Name:       INTRFC_E.VHD
-- Host Machine:    GATEWAY 486/33 (IBM AT Clone)
-- Target Machine:  GATEWAY 486/33 (IBM AT Clone)
-- Environment :    Model Technology VHDL Simulation for Windows (Ver 4.3f)
--                  DOS Version 6.2
-- Organization:    NASA-LaRC
-- Project:         Fly By Light - Power By Wire (FBL-PBW)
-- Author:          Mahyar R. Malekpour
-- Creation Date:   10/24/95
--------------------------------------------------------------------------------
-- Name/Number:
--    INTRFC                                (entity)
--
-- Abstract:
--   This file contains the entity declaration for the PC interface that will
-- be programmed on a Xilinx XC3000.
--
--   SIGNAL DEFINITION :
--
--
-- Acronyms/Abbreviations:
--   FBL/PBW
--   BIU - Bus Interface Unit
--
-- Dependencies:
--   IEEE.STD_LOGIC_1164
--
-- Global Objects:
--
-- Exceptions:
--
-- Machine/Compiler Dependencies:
--
-- Revisions:
--
--   Modified on:  10/24/95
--   by:           Mahyar Malekpour
--
--   1.  A head count of the I/O pins (as of now):
-- 41 I/O pins needed:
--      18 for two FIFOs, 8 data lines, 10 address lines, 3 control lines
--      (AEN, IOWR, and IORD), and 2 from the two FIFOs.
-- XC300 provides us with 54 I/O pins,
-- Therefore, our PC interface should fit inside one XC3000, but the BIU
-- will not!  The BIU will require, at least, 18 I/O pins for the two FIFOs
-- interface in addition to its other I/O pins.
--
--   Modified on:  3/12/96
--   by:           Mahyar Malekpour
--   1.  Bring in the half-full flags of the FIFOs.  Two more I/O pins used.
--
--   Modified on:  3/19/96
--   by:           Mahyar Malekpour
--
--   Modified on:  6/7/1996
--   by:           Mahyar Malekpour
-- Modified the entities to reflect the newly added Direction signal.
--
--------------------------------------------------------------------------------

library IEEE ;
use IEEE.std_logic_1164.all ;
use WORK.CNSTNT_P.all ;

entity INTRFC is
    port (
-- the following 4 signals are not synthesizable and so are commented out.
-- 3/19/96
--           RESET                : in    std_logic ; -- Power on reset, active high
--           CLK_In               : in    std_logic ;
--           Serial_Prog_In       : in    std_logic ;
--           Done_Prog_Bar        : in    std_logic ;

           Reset_BIU            : out   std_logic ;
           Direction            : out   std_logic ;
```

```
            Data_Read_Bar         : out   std_logic ;
            Data_Write_Bar        : out   std_logic ;

            Chip_Select_Bar       : in    std_logic ; -- chip select, active low

            ADDRESS               : in    std_logic_vector (PORT_Length downto 0) ;
            IOWR_Bar              : in    std_logic ;
            IORD_Bar              : in    std_logic ;

            uP_Data_In_Out        : inout std_logic_vector (Data_Length downto 0) ;
            FIFO_Data_In_Out      : inout std_logic_vector (Data_Length downto 0) ;

            Input_FIFO_Reset_Bar  : out   std_logic ;
            Input_FIFO_Read_Bar   : out   std_logic ;
            Input_FIFO_Write_Bar  : out   std_logic ;
            Input_FIFO_Full_Bar   : in    std_logic ;
            Input_FIFO_Empty_Bar  : in    std_logic ;
            Input_FIFO_HF_Bar     : in    std_logic ;

            Output_FIFO_Reset_Bar : out   std_logic ;
            Output_FIFO_Read_Bar  : out   std_logic ;
            Output_FIFO_Write_Bar : out   std_logic ;
            Output_FIFO_Full_Bar  : in    std_logic ;
            Output_FIFO_Empty_Bar : in    std_logic ;
            Output_FIFO_HF_Bar    : in    std_logic ;

            BIU_FIFO_Read_Bar     : in    std_logic ;
            BIU_FIFO_Write_Bar    : in    std_logic ;

            CCLK_4000             : out   std_logic ;
            Prog_4000             : out   std_logic ;
            INIT_4000             : in    std_logic ;
            DONE_4000             : in    std_logic

          ) ;

end INTRFC ;

--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
-- File Name:       INTRFC_A.VHD
-- Host Machine:    GATEWAY 486/33 (IBM AT Clone)
-- Target Machine:  GATEWAY 486/33 (IBM AT Clone)
-- Environment :    Model Technology VHDL Simulation for Windows (Ver 4.3f)
--                  DOS Version 6.2
-- Organization:    NASA-LaRC
-- Project:         Fly By Light - Power By Wire (FBL-PBW)
-- Author:          Mahyar R. Malekpour
-- Creation Date:   10/24/95
--------------------------------------------------------------------------------
-- Name/Number:
--    INTRFC                                (architecture)
--
-- Abstract:
--   This file contains the architecture for the PC interface that will
-- be programmed on a Xilinx XC3000.
--
--   SIGNAL DEFINITION :
--
--
-- Acronyms/Abbreviations:
--   FBL/PBW
--   BIU - Bus Interface Unit
--
-- Dependencies:
--   IEEE.STD_LOGIC_1164
--   WORK.CNSTNT_P.all
--
-- Global Objects:
--
-- Exceptions:
--
-- Machine/Compiler Dependencies:
--
-- Revisions:
```

```
--
--   Modified on:  11/22/95
--   by:           Mahyar Malekpour
--
--   Declared a constant, Xilinx_Delay, to reflect and study the effect of
-- inherent propagation delay in the Xilinx 3000.  The preliminary resuts
-- indicate that we may have some timing problem while fetching, i.e.
-- reading, data from the Output_FIFO.
--   Xilinx_Delay = 30   ===> it works
--   Xilinx_Delay = 40   ===> it doesn't work
-- Need to study this futher.
--
--   Modified on:  11/27/95
--   by:           Mahyar Malekpour
--
--   Modified the code to overcome the timing problems associated with the
-- propagation delay imposed by the Xilinx FPGA.  While reading data from
-- the FIFO, the bidirectional bus is now controlled directly by the
-- IORD_Bar signal.  With this arrangement, the data bus will not be driven
-- by the FIFOs when the bus is to be tri-stated.   The time period where the
-- IORD_Bar is active, when low, has to be long enogh to account for the
-- Xilinx delay as well as FIFO response time.  This time is about >= 70 ns.
--
--   Modified on:  11/30/95
--   by:           Mahyar Malekpour
--
--   Modified the BIU_Read_Write_FIFO process to emulate activities of the
-- BIUs.  See notes by the BIU_Read_Write_FIFO process.
--
--   Modified on:  3/12/96
--   by:           Mahyar Malekpour
--
--
--   Modified on:  3/28/96
--   by:           Mahyar Malekpour
--
--  Added 4 new signals to this module:
--        Reset_BIU, uP_Data_Pin_5, uP_Data_Pin_6, uP_Data_Pin_7
-- These signal are send out for possible future use in other parts of the
-- BIU.  With this addition, all bits of the uP data bus are used when the
-- uP addresses the "reset" port.  Thus, the uP can, thru software,
-- selectively and/or collectively reset parts or all of the system.
--
-- uP_Data_In_Out (0) : reset input FIFO
-- uP_Data_In_Out (1) : reset output FIFO
-- uP_Data_In_Out (2) : input FIFO write and output FIFO read
-- uP_Data_In_Out (3) : output FIFO write and input FIFO read
-- uP_Data_In_Out (4) : reset BIU, i.e., global reset
-- uP_Data_In_Out (5) : Not used
-- uP_Data_In_Out (6) : Not used
-- uP_Data_In_Out (7) : Not used
--
--   Modified on:  4/1/96
--   by:           Mahyar Malekpour
--
--  Latched the reset commands written to the Command_ADDRESS from the uP
-- data bus into an internal register, Latched_Command, for futher use.
--
--   Modified on:  4/16/96
--   by:           Mahyar Malekpour
--  The bit 4 of the latched command is used as a global reset to the BIU,
-- and hence is named Reset_BIU.  When high, the BIU and the FIFOs are reset.
-- Since it is latched, it must be lowered after some time interval for the
-- normal operations to resume.
--
--   Modified on:  4/19/96
--   by:           Mahyar Malekpour
-- Simplified the code and got rid of the previous modifications.
--
--   Modified on:  5/2/96
--   by:           Mahyar Malekpour
-- Added the last segment to the bus so data can be written to and read from
-- the XC4000 via the same bus that is used to access the two FIFOs.   Thus,
-- this bus, FIFO_Data_In_Out bus, is being driven from five directions and
-- thru three ports.  This bus is also used to program the XC4000 via the
-- XC3020 in the parallel synchronous prepheral mode.
```

```
--
--      a. Data port:
--              1. read FIFO_Out
--              2. write FIFO_In
--      b. PROG_4000 port
--              3. program XC4000
--      c. 4000_Status port
--              4. read BIU status
--              5. write schedule to RAM
--
--
--   Modified on:  6/7/1996
--   by:           Mahyar Malekpour
-- Modified the entities to reflect the newly added Direction signal.
--
--   Modified on:  8/27/96
--   by:           Mahyar Malekpour
--
--   Once again there is a need to individually reset the FIFOs and
-- independely from the BIU.  Therefore, using the same old "reset" port
-- the FIFOs and the BIU can now be reset thru the following data bits:
--
-- uP_Data_In_Out (0) : reset input FIFO
-- uP_Data_In_Out (1) : reset output FIFO
-- uP_Data_In_Out (2) : Not used
-- uP_Data_In_Out (3) : Not used
-- uP_Data_In_Out (4) : reset BIU
-- uP_Data_In_Out (5) : Not used
-- uP_Data_In_Out (6) : Not used
-- uP_Data_In_Out (7) : Not used
--
--   Thus, there is no such thing as global reset anymore.
--
--   Note: I have also inverted the Reset_BIU input, i.e., data bit 4, to
-- be consistent with the FIFO reset input bits.  MUST reflect this change
-- in the C/C++ code of the test-bench.
--
--
--------------------------------------------------------------------------------

library IEEE ;
use IEEE.std_logic_1164.all ;
use WORK.CNSTNT_P.all ;


architecture INTRFC_Behaviour of INTRFC is

    ------------------------------------------------------------------------

    signal   Latched_Command : std_logic_vector (Data_Length downto 0) ;

    signal      Data_SELECTED_Bar : std_logic ;
    signal              FIFO_Read : std_logic ;

    signal    Status_SELECTED_Bar : std_logic ;
    signal            Status_Read : std_logic ;

    signal   Command_SELECTED_Bar : std_logic ;
    signal PROG_4000_SELECTED_Bar : std_logic ;
    signal         PROG_4000_Read : std_logic ;

begin

    ------------------------------------------------------------------------
    -- This process decifers the incomming address bits and activates one of
    -- the selected ports used in this module.
    --
    Check_Addresses : process ( Chip_Select_Bar, ADDRESS )

    begin

      Command_SELECTED_Bar <= '1' ;
      Data_SELECTED_Bar <= '1' ;
      Status_SELECTED_Bar <= '1' ;
      PROG_4000_SELECTED_Bar <= '1' ;
```

```
  if ( ADDRESS = Data_ADDRESS ) and ( Chip_Select_Bar = '0' ) then
    -- Writing data to the FIFO_In and reading data from the FIFO_Out
    Data_SELECTED_Bar <= '0' ;
  elsif ( ADDRESS = Status_ADDRESS ) and ( Chip_Select_Bar = '0' ) then
    -- reading the FIFO-staus-register contents
    Status_SELECTED_Bar <= '0' ;
  elsif ( ADDRESS = Command_ADDRESS ) and ( Chip_Select_Bar = '0' ) then
    -- Resetting the BIU and the FIFOs
    Command_SELECTED_Bar <= '0' ;
  elsif ( ADDRESS = PROG_4000_ADDRESS ) and ( Chip_Select_Bar = '0' ) then
    -- reseting and programming the XC4000 and reading XC4000 status
    -- from the INIT and DONE signals.  All thru the same port.
    PROG_4000_SELECTED_Bar <= '0' ;

  end if ;

end process ;

-------------------------------------------------------------------------
-- Defining a bi-deirectional buffer for the data bus
-- The next two processes work together to define the bi-deirectional bus
--
uP_Read_FIFO_Status : process ( FIFO_Read, Status_Read, PROG_4000_Read,
                          FIFO_Data_In_Out, INIT_4000, DONE_4000,
                          Input_FIFO_Full_Bar, Input_FIFO_Empty_Bar, Input_FIFO_HF_Bar,
                          Output_FIFO_Full_Bar, Output_FIFO_Empty_Bar, Output_FIFO_HF_Bar )
begin
    if ( FIFO_Read = '0' ) then
      uP_Data_In_Out <= FIFO_Data_In_Out ;
    elsif ( Status_Read = '0' ) then
      uP_Data_In_Out (0) <= Input_FIFO_Full_Bar ;
      uP_Data_In_Out (1) <= Input_FIFO_Empty_Bar ;
      uP_Data_In_Out (2) <= Input_FIFO_HF_Bar ;
      uP_Data_In_Out (3) <= Output_FIFO_Full_Bar ;
      uP_Data_In_Out (4) <= Output_FIFO_Empty_Bar ;
      uP_Data_In_Out (5) <= Output_FIFO_HF_Bar ;
      uP_Data_In_Out (6) <= '0' ;
      uP_Data_In_Out (7) <= '0' ;
   elsif ( PROG_4000_Read = '0' ) then
      uP_Data_In_Out (0) <= INIT_4000 ;
      uP_Data_In_Out (1) <= DONE_4000 ;
      uP_Data_In_Out (2) <= '0' ;
      uP_Data_In_Out (3) <= '0' ;
      uP_Data_In_Out (4) <= '0' ;
      uP_Data_In_Out (5) <= '0' ;
      uP_Data_In_Out (6) <= '0' ;
      uP_Data_In_Out (7) <= '0' ;
     else
      uP_Data_In_Out <= "ZZZZZZZZ" ;
    end if ;

end process ;

-------------------------------------------------------------------------
uP_Write_FIFO : process ( Data_SELECTED_Bar, uP_Data_In_Out,
                          PROG_4000_SELECTED_Bar )
begin
    if (( Data_SELECTED_Bar = '0' ) or ( PROG_4000_SELECTED_Bar = '0' )) then
      FIFO_Data_In_Out <= uP_Data_In_Out ;
    else
      FIFO_Data_In_Out <= "ZZZZZZZZ" ;
    end if ;

end process ;

-------------------------------------------------------------------------
Latch_Command : process ( Command_SELECTED_Bar, uP_Data_In_Out, IOWR_Bar )
begin
  if ( Command_SELECTED_Bar = '0' ) and ( IOWR_Bar = '0' ) then  -- Latch in the uP data bus.
    Latched_Command <= uP_Data_In_Out ;
  end if ;

end process ;

-------------------------------------------------------------------------
 -- Time to reset the BIU
```

```
    Reset_BIU <= not Latched_Command (4) ;

    -- The buffer should transfer data to the uP data bus whenever uP attempts
    -- to read any port in the XC3020 and xc4000.
    -- 6/7/1996 Mahyar Malekpour
    --
    Direction <= IORD_Bar or Chip_Select_Bar ;

    -- Time to reset the FIFOs
    -- 8/27/96
    Input_FIFO_Reset_Bar  <= Latched_Command (0) ;
    Output_FIFO_Reset_Bar <= Latched_Command (1) ;

    -- Send the rest of the data bus out for future use in other
    -- parts of the BIU.
    -- Mahyar 3/28/1996
    Prog_4000 <= Latched_Command (5) ;
    Data_Read_Bar  <= Latched_Command (6) ;
    Data_Write_Bar <= Latched_Command (7) ;

    -- Time to read or write to the FIFOs
    FIFO_Read   <= IORD_Bar or Data_SELECTED_Bar ;

    -- Time to read the FIFOs status
    Status_Read <= IORD_Bar or Status_SELECTED_Bar ;

    -- Time to grogram the XC4000
    PROG_4000_Read  <= IORD_Bar or PROG_4000_SELECTED_Bar ;
    CCLK_4000       <= IOWR_Bar or PROG_4000_SELECTED_Bar ;

    ------------------------------------------------------------------------
    -- This process processes all selected commands for reading and writing
    -- to the I/O ports.  It also initializes the signals at the power on.
    --
    Command_Process : process ( Data_SELECTED_Bar, Latched_Command,
                                IORD_Bar, IOWR_Bar, Status_SELECTED_Bar,
                                PROG_4000_SELECTED_Bar )

    begin

      -- Time to reset the system
--      if ( Latched_Command (4) = '0' ) then  -- i.e. if Reset_BIU is low
-- 8/27/96
      if ( Latched_Command (0) = '0' ) then  -- i.e. if reset input FIFO
        -- The following 2 signals need to be high during the reset process
        -- therefore, they can be tied and controled by a single data line.
        Input_FIFO_Write_Bar  <= '1' ;

      elsif ( Latched_Command (1) = '0' ) then  -- i.e. if reset output FIFO
        Output_FIFO_Read_Bar  <= '1' ;

      else
        Input_FIFO_Write_Bar <= IOWR_Bar or Data_SELECTED_Bar ;
        Output_FIFO_Read_Bar <= IORD_Bar or Data_SELECTED_Bar ;

      end if ;
    end process ;

    ------------------------------------------------------------------------
    BIU_Read_Write_FIFO : process ( Latched_Command,
                                    BIU_FIFO_Write_Bar, BIU_FIFO_Read_Bar )
    begin

--      if ( Latched_Command (4) = '0' ) then  -- i.e. if Reset_BIU is low
-- 8/27/96
      if ( Latched_Command (0) = '0' ) then  -- i.e. if reset input FIFO
        -- The following 2 signals need to be high during the reset process
        -- therefore, they can be tied and controled by a single data line.
        Input_FIFO_Read_Bar   <= '1' ;

      elsif ( Latched_Command (1) = '0' ) then  -- i.e. if reset output FIFO
        Output_FIFO_Write_Bar <= '1' ;

      else
        -- BIU is reading from the Input_FIFO
        Input_FIFO_Read_Bar <= BIU_FIFO_Read_Bar ;
```

```
        -- BIU is writing to the Output_FIFO
        Output_FIFO_Write_Bar <= BIU_FIFO_Write_Bar ;

      end if ;

    end process ;

    -------------------------------------------------------------------------

end INTRFC_Behaviour ;

-------------------------------------------------------------------------------
-------------------------------------------------------------------------------
-------------------------------------------------------------------------------
-- File Name:       uP_PRT_E.VHD
-- Host Machine:    GATEWAY 486/33 (IBM AT Clone)
-- Target Machine:  GATEWAY 486/33 (IBM AT Clone)
-- Environment :    Model Technology VHDL Simulation for Windows (Ver 4.3f)
--                  DOS Version 6.2
-- Organization:    NASA-LaRC
-- Project:         Fly By Light - Power By Wire (FBL-PBW)
-- Author:          Mahyar R. Malekpour
-- Creation Date:   3/19/96
-------------------------------------------------------------------------------
-- Name/Number:
--    uP_PRT                                (entity)
--
-- Abstract:
--   This file contains the entity declaration for the interface part of
-- the PC.  It simply decodes the address, using only the upper bits, and
-- generats a chip select signal to activate the BIU and uP interactions.
-- This entity is to be programmed on a Xilinx XC3020.
--
--   SIGNAL DEFINITION :
--
--
-- Acronyms/Abbreviations:
--   FBL/PBW
--   BIU - Bus Interface Unit
--
-- Dependencies:
--   IEEE.STD_LOGIC_1164
--
-- Global Objects:
--
-- Exceptions:
--
-- Machine/Compiler Dependencies:
--
-- Revisions:
--
--   Modified on:  ??/??/1996
--   by:           Mahyar Malekpour
--
-------------------------------------------------------------------------------

library IEEE ;
use IEEE.std_logic_1164.all ;
use WORK.CNSTNT_P.all ;

entity uP_PRT is
    port (
           ADDRESS          : in    std_logic_vector (9 downto 0) ;
           AEN_Bar          : in    std_logic ; -- Address enable, active high
           Chip_Select_Bar  : out   std_logic   -- chip select, active low

         ) ;

end uP_PRT ;

-------------------------------------------------------------------------------
-------------------------------------------------------------------------------
-------------------------------------------------------------------------------
-- File Name:       uP_PRT_A.VHD
-- Host Machine:    GATEWAY 486/33 (IBM AT Clone)
```

```
-- Target Machine:  GATEWAY 486/33 (IBM AT Clone)
-- Environment :    Model Technology VHDL Simulation for Windows (Ver 4.3f)
--                  DOS Version 6.2
-- Organization:    NASA-LaRC
-- Project:         Fly By Light - Power By Wire (FBL-PBW)
-- Author:          Mahyar R. Malekpour
-- Creation Date:   3/19/96
--------------------------------------------------------------------------------
-- Name/Number:
--    uP_PRT                                (architeture)
--
-- Abstract:
--   This file contains the entity declaration for the interface part of
-- the PC.  It simply decodes the address, using only the upper bits, and
-- generates a chip select signal to activate the BIU and uP interactions.
-- This entity is to be programmed on a Xilinx XC3020.
--
--   SIGNAL DEFINITION :
--
--
-- Acronyms/Abbreviations:
--   FBL/PBW
--   BIU - Bus Interface Unit
--
-- Dependencies:
--   IEEE.STD_LOGIC_1164
--
--   CNSTNT_P
--
-- Global Objects:
--
-- Exceptions:
--
-- Machine/Compiler Dependencies:
--
-- Revisions:
--
--   Modified on:  ??/??/??
--   by:           Mahyar Malekpour
--
--------------------------------------------------------------------------------

library IEEE ;
use IEEE.std_logic_1164.all ;
use WORK.CNSTNT_P.all ;

architecture uP_PRT_Behave of uP_PRT is

begin

    Check_Addresses : process ( ADDRESS, AEN_Bar )
    begin
      if ( ADDRESS (9 downto (PORT_Length + 1)) = BASE_ADDRESS ) then
        Chip_Select_Bar <= AEN_Bar ;
      else
        Chip_Select_Bar <= '1' ;
      end if ;

    end process ;

    --------------------------------------------------------------------------

end uP_PRT_Behave ;

--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
-- File Name:       XC4005_E.VHD
-- Host Machine:    GATEWAY 486/33 (IBM AT Clone)
-- Target Machine:  GATEWAY 486/33 (IBM AT Clone)
-- Environment :    Model Technology VHDL Simulation for Windows (Ver 4.3f)
--                  DOS Version 6.2
-- Organization:    NASA-LaRC
-- Project:         Fly By Light - Power By Wire (FBL-PBW)
-- Author:          Mahyar R. Malekpour
-- Creation Date:   6/10/1996 based on xc4000 created on 03/22/96
```

```
--------------------------------------------------------------------------------
-- Name/Number:
--    XC4005                                (entity)
--
-- Abstract:
--   This file contains the entity declaration for the FBL/PBW fault-tolerant
--   architecture BIU.
--
--   SIGNAL DEFINITION :
--
--   BIU_ID            : BIU ID
--   S_In              : Serial data into BIU
--   S_Out             : Serial data out of BIU
--   Reset_BIU         : asserted by peripheral microprocessor to reset BIU, active high
--                       This signal is active high because the Flip-Flops in
--                     Xilinx 4000 require high for "clr".
--   FIFO_Data_In      : 8-bit data from FIFO_In into BIU
--   FIFO_Read_Bar     : triggers reading from the input FIFO, FIFO_In, active low
--   FIFO_Data_Out     : 8-bit data out of BIU to FIFO_Out
--   FIFO_Write_Bar    : triggers writing to the output FIFO, FIFO_Out, active low
--   Chip_Select_Bar   : active low, used by uProcessor to select BIU for
--                     uProcessor access.
--   BIU_Reset         : reset line to BIU from uProcessor, active low
--   ROM_Data          : data lines from EPROM
--   ROM_Read_bar      : active low, read line to EPROM.
--   ROM_ADDRESS       : address lines to EPROM
--   Clock_In          : Input clock to the BIU = Bit clock
--   Clock_Out         : Output clock of the BIU = Byte clock
--
-- Acronyms/Abbreviations:
--   FBL/PBW
--   BIU - Bus Interface Unit
--
-- Dependencies:
--   IEEE.STD_LOGIC_1164
--
-- Global Objects:
--
-- Exceptions:
--
-- Machine/Compiler Dependencies:
--
-- Revisions:
--
--   Modified on:  3/27/96
--   by:           Mahyar Malekpour
--
--   Added "Reset_BIU" signal to clear the 4-bit counter and reset it in a
-- known initial state.  This signal is active high because the Flip-Flops
-- in Xilinx 4000 require high for "clr".
--
--   Modified on:  4/12/96
--   by:           Mahyar Malekpour
-- Added three address lines so that I can transfer data between the FIFOs
-- on demand.  Again, this is for intermediate step and for test purposes.
-- It will have to be modified later.
--
--   Modified on:  5/6/96
--   by:           Mahyar Malekpour
-- Added the last segment of the FIFO Bus, FIFO_Data_In_Out, that connects
-- XC4000 to XC3020 and thus to the uP.  It allows read and write of data
-- to and from the XC4000 status registers as well as the adjoining RAM
-- that holds the schedule of events.  This segment of the bus MUST be
-- tri-stated when not in use as other segments are.  Two new signals,
-- Data_Read_Bar and Data_Write_Bar, were also added for the correspong
-- operation to be controlled by the uP.
--
--   Modified on:  8/22/96
--   by:           Mahyar Malekpour
-- Added Switch_Time_In and INTRPT_Out to this entity.
-- Switch_Time_In is provided to give the user more control over the switch time.
-- INTRPT_Out is used to let the uP know a new cycle started.
--
--   Modified on:  8/27/96
--   by:           Mahyar Malekpour
-- Added INTRPT_ACK_In to this entity.
```

```
-- INTRPT_ACK_In is used to let the BIU know that the uP has serviced the
-- interrupt.
--
--   Modified on:  9/4/96
--   by:           Mahyar Malekpour
-- Added MUX_Select to this entity.
--
--
--------------------------------------------------------------------------------

library IEEE ;
use IEEE.std_logic_1164.all ;
use WORK.CNSTNT_P.all ;
use WORK.EPROM_P.all ;

entity XC4005 is
    PORT (
           BIU_OR_RMU            : in    std_logic ;
           BIU_ID                : in    std_logic_vector (Data_Length downto 0) ;
--           S_In                  : in    std_logic ;
--           S_Out                 : out   std_logic ;

           Reset_BIU             : in    std_logic ;
           Chip_Select_Bar       : in    std_logic ;

           FIFO_Data_In          : in    std_logic_vector (Data_Length downto 0) ;
           Input_FIFO_Read_Bar   : out   std_logic ;
           Input_FIFO_Full_Bar   : in    std_logic ;
           Input_FIFO_Empty_Bar  : in    std_logic ;
           Input_FIFO_HF_Bar     : in    std_logic ;

           FIFO_Data_Out         : out   std_logic_vector (Data_Length downto 0) ;
           Output_FIFO_Write_Bar : out   std_logic ;
           Output_FIFO_Full_Bar  : in    std_logic ;
           Output_FIFO_Empty_Bar : in    std_logic ;
           Output_FIFO_HF_Bar    : in    std_logic ;

           FIFO_Data_In_Out      : inout std_logic_vector (Data_Length downto 0) ;
           Data_Read_Bar         : in    std_logic ;
           Data_Write_Bar        : in    std_logic ;

           Bit_Clock_In          : in    std_logic ;
           Byte_Clock_Out        : out   std_logic ;

           ADDRESS               : in    std_logic_vector (PORT_Length downto 0) ;
           IOWR_Bar              : in    std_logic ;
           IORD_Bar              : in    std_logic ;

           Serial_Data_In        : in    std_logic ;
           Serial_Data_Out       : out   std_logic ;

           -- the folowing signals are for test purposes only
           -- get rid of them later
           Latched_Sync_Out : out   std_logic ;

           ROM_Data              : in    std_logic_vector (ROM_WIDTH - 1 downto 0) ;
           ROM_Read_Bar          : out   std_Logic ; -- := '0' ;  -- active low
           ROM_Write_Bar         : out   std_Logic ; -- := '0' ;  -- active low
           ROM_ADDRESS           : out   std_logic_vector (ROM_ADDRESS_LINES - 1 downto 0) ;

           Switch_Time_In        : in    std_logic_vector ( 2 downto 0 ) ; -- three bits for now
           INTRPT_Out            : out   std_logic ;
           INTRPT_ACK_In         : in    std_logic ;

           MUX_Select            : out   std_logic_vector (Data_Length downto 0)

         ) ;

end XC4005 ;

--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
-- File Name:       XC4005_A.VHD
-- Host Machine:    GATEWAY 486/33 (IBM AT Clone)
-- Target Machine:  GATEWAY 486/33 (IBM AT Clone)
```

```
-- Environment :    Model Technology VHDL Simulation for Windows (Ver 4.3f)
--                  DOS Version 6.2
-- Organization:    NASA-LaRC
-- Project:         Fly By Light - Power By Wire (FBL-PBW)
-- Author:          Mahyar R. Malekpour
-- Creation Date:   6/10/1996 based on xc4000 created on 03/22/96
-------------------------------------------------------------------------------
-- Name/Number:
--    XC4005                                (architecture)
--
-- Abstract:
--   This file contains the architecture for the FBL/PBW fault-tolerant
--   architecture BIU.
--
--   SIGNAL DEFINITION :
--
--   BIU_ID            : BIU ID
--   S_In              : Serial data into BIU
--   S_Out             : Serial data out of BIU
--   FIFO_Data_In      : 8-bit data from FIFO_In into BIU
--   FIFO_Read_Bar     : triggers reading from the input FIFO, FIFO_In, active low
--   FIFO_Data_Out     : 8-bit data out of BIU to FIFO_Out
--   FIFO_Write_Bar    : triggers writing to the output FIFO, FIFO_Out, active low
--   Chip_Select_Bar   : active low, used by uProcessor to select BIU for uProcessor access.
--   BIU_Reset         : reset line to BIU from uProcessor, active ?
--   ROM_Data          : data lines from EPROM
--   ROM_Read_bar      : active low, read line to EPROM.
--   ROM_ADDRESS       : address lines to EPROM
--   Bit_Clock_In      : Input clock to the BIU = Bit clock
--   Byte_Clock_Out    : Output clock of the BIU = Byte clock
--
-- Acronyms/Abbreviations:
--   BIU - Bus Interface Unit
--
-- Dependencies:
--    IEEE.STD_LOGIC_1164
--
-- Global Objects:
--
-- Exceptions:
--
-- Machine/Compiler Dependencies:
--
-- Revisions:
--
--   Modified on:  5/20/96
--   by:           Mahyar Malekpour
--  Added "Strobe" signal that is used to load in data from the FIFO_In into
-- the p-to-s register.  "Strobe" is active only for one bit clock cycle.
--
--   Modified on:  6/11/96
--   by:           Mahyar Malekpour
--  Added P_to_S and S_to_P components.
--
--   Modified on:  6/12/96
--   by:           Mahyar Malekpour
--  Added PSCON component.
--
--   Modified on:  6/17/96
--   by:           Mahyar Malekpour
-- P_to_S entity operates on the NEGATIVE edge of the Bit_Clock,
-- S_to_P entity operates on the POSITIVE edge of the Bit_Clock,
-- and everything else operate on the POSITIVE edge of the Byte_Clock.
--
--   Modified on:  8/1/96
--   by:           Mahyar Malekpour
--
-- Read_Data_Count added to read the first three bytes, input data packet header,
-- from the input FIFO.  These three bytes are FF, BIU_ID, and Count respectively.
-- See notes in DATCLK_A.VHD file.
--
--   Modified on:  8/2/96
--   by:           Mahyar Malekpour
--
-- Registering the errors encountered in the designated bit position of the
-- Status_Reg_0.
```

```
--
-- Register Status_Reg_1 is put a side for the micro-processor to write whatever
-- seemed necessay.
--
--   Modified on:  8/7/96
--   by:           Mahyar Malekpour
-- Added Command_Data_Flag to make the most of the S_to_P_Count.  The S_to_P_Count
-- is now treated as the command register while Command_Data_Flag is set high
-- and as a data byte counter when Command_Data_Flag is set low.
--
--   Modified on:  8/8/96
--   by:           Mahyar Malekpour
--   Added four temporary buffers, Temp_1_Buffer thru Temp_4_Buffer, so that
-- the first three bytes of the data packet header will be around for the next
-- three Byte clocks.  It is essential to keep these header bytes around for
-- sending them to the output FIFO after matching the BIU_ID.
--   As a result, will have to FLUSH these buffers so that all of the incomming
-- data bytes are transfered to the output FIFO.   Thus, the total count is
-- +2 more than the number of data bytes in the incomming data packet.
-- Therefore, I increased the size of the S_to_P_Count counter by one bit to
-- accomodate for the extra two dat bytes.
--
--   Modified on:  8/15/96
--   by:           Mahyar Malekpour
-- Separating the Byte_Clock to handel the Sync_Detected.
--
--   Modified on:  8/22/96
--   by:           Mahyar Malekpour
-- Added Switch_Time_In and INTRPT_Out to this entity.
-- Switch_Time_In is provided to give the user more control on the swutch time.
-- INTRPT_Out is used to let the uP know a new cycle started.
--
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------

library IEEE ;
--use IEEE.std_logic_1164.all ;
use WORK.CNSTNT_P.all ;
use ieee.std_logic_arith.all ;
--use ieee.std_logic_signed.CONV_INTEGER ;
use work.my_std_logic_arith.all ;


architecture XC4005_Behave of XC4005 is

    ------------------------------------------------------------------------
    component BYTCLK
    port ( Reset_BIU               : in    std_logic ;
           Start_Cycle             : in    std_logic ;
           Sync_Detected           : in    std_logic ;
           Bit_Clock_In            : in    std_logic ;
           Fixed_Byte_Clock_Out    : out   std_logic ;
           Strobe_Out              : out   std_logic ;
           Variable_Byte_Clock_Out : out   std_logic
         ) ;

    end component ;

    component S_to_P
    PORT (
           Bit_Clock_In          : in    std_logic ;
           Serial_Data_In        : in    std_logic ;
           Parallel_Data_Out     : out   std_logic_vector ((Data_Length_Plus_1) downto 0) ;
           Mode_Bit_Out          : out   std_logic

         ) ;

    end component ;

    component P_to_S
    PORT (
           Bit_Clock_In        : in    std_logic ;
           Parallel_Data_In    : in    std_logic_vector (Data_Length downto 0) ;
           Mode_Bit_In         : in    std_logic ;
           Load_Parallel       : in    std_logic ;
           Serial_Data_Out     : out   std_logic
```

```
     ) ;

end component ;

component PSCON
port (
     Load_P_TO_S_Count : in    STD_LOGIC ;
     Count_Value       : in    std_logic_vector (Data_Length downto 0) ;
     FIFO_Empty_Bar    : in    STD_LOGIC ;
     Bit_Clock         : in    STD_LOGIC ;
     BYTE_CLOCK        : in    STD_LOGIC ;
     Read_FIFO_Error   : out   std_logic ;
     FIFO_RD_bar       : out   STD_LOGIC ) ;

end component ;

component HEADER
port (
       BIU_OR_RMU         : in    std_logic ;
       Reset_BIU          : in    std_logic ;
       BIU_ID             : in    std_logic_vector ( Data_Length downto 0 ) ;
       Mode_Bit_In        : in    std_logic ;
       Data_In            : in    std_logic_vector ( Data_Length_Plus_1 downto 0 ) ;
       Byte_Clock_In      : in    std_logic ;
       Sync_Detected_Out  : out   std_logic ;
       Command_Data_Flag  : out   std_logic ;
       Load_Counter_Out   : out   std_logic
) ;

end component ;

component DATCLK
port ( Reset_BIU       : in    std_logic ;
       Transmit_Data   : in    std_logic ;
       Byte_Clock_In   : in    std_logic ;
       Count_Value_Out : out   std_logic_vector ( 1 downto 0 )
) ;
end component ;

component PRMCON
port (
       BIU_OR_RMU       : in    std_logic ;
       Reset_BIU        : in    std_logic ;
       Sync_Detected    : in    std_logic ;
       Start_Cycle      : in    std_logic ;
       BIU_ID           : in    std_logic_vector ( Data_Length downto 0 ) ;
       Byte_Clock_In    : in    std_logic ;
       Start_Transmit   : out   std_logic ;
       Start_Receieve   : out   std_logic ;
       Status_Data      : out   std_logic ;
       Start_Command    : out   std_logic ;
       MUX_Select       : out   std_logic_vector (Data_Length downto 0) ;

       ROM_Data         : in    std_logic_vector (ROM_WIDTH - 1 downto 0) ;
       ROM_Read_Bar     : out   std_Logic ; -- := '0' ;  -- active low
       ROM_Write_Bar    : out   std_Logic ; -- := '0' ;  -- active low
       ROM_ADDRESS      : out   std_logic_vector (ROM_ADDRESS_LINES - 1 downto 0) ;
       EPROM_Error_Flag : out   std_logic
) ;
end component ;

component RECEVR
port ( Reset_BIU        : in    std_logic ;
       Start_Cycle      : in    std_logic ;
       Receieve_Data    : in    std_logic ;
       Byte_Clock_In    : in    std_logic ;
       Load_Command_Reg : in    std_logic ;
       Start_Receieve   : out   std_logic ;
       Receive_Error_1  : out   std_logic ;
       Receive_Error_2  : out   std_logic ;
       Switch_Time_In   : in    std_logic_vector ( 2 downto 0 )  -- three bits for now
) ;

end component ;
```

```
    component STATUS
    port (
           BIU_OR_RMU       : in    std_logic ;
           Reset_BIU        : in    std_logic ;
           Start_Cycle      : in    std_logic ;
           BIU_ID           : in    std_logic_vector ( Data_Length downto 0 ) ;
           Start_Command    : in    std_logic ;
           Start_Transmit   : in    std_logic ;
           Data_Status_Flag : in    std_logic ;
           Data_Mode_Bit    : in    std_logic ;
           FIFO_Data_In     : in    std_logic_vector ( Data_Length downto 0 ) ;

           Byte_Clock_In    : in    std_logic ;
           Status_Reg_In    : in    std_logic_vector ( Data_Length downto 0 ) ;

           Transmit_Data    : out   std_logic ;
           Load_Byte_Out    : out   std_logic ;
           Mode_Bit_Out     : out   std_logic ;
           Data_Status_Out  : out   std_logic_vector ( Data_Length downto 0 )
    ) ;
    end component ;

    -------------------------------------------------------------------------
    -------------------------------------------------------------------------
    for all : S_to_P use entity work.S_to_P (S_to_P_Behave) ;
    for all : P_to_S use entity work.P_to_S (P_to_S_Behave) ;
    for all : PSCON  use entity work.PSCON (PSCON_Behave) ;
    for all : BYTCLK use entity work.BYTCLK (BYTCLK_Behave) ;
    for all : HEADER  use entity work.HEADER ( HEADER_Behave ) ;
    for all : DATCLK  use entity work.DATCLK ( DATCLK_Behave ) ;
    for all : PRMCON  use entity work.PRMCON ( PRMCON_Behave ) ;
    for all : RECEVR  use entity work.RECEVR ( RECEVR_Behave ) ;
    for all : STATUS  use entity work.STATUS ( STATUS_Behave ) ;

    -------------------------------------------------------------------------
    -------------------------------------------------------------------------
    signal Output_Data_Buffer : std_logic_vector (Data_Length_Plus_1 downto 0) ;
    signal      Temp_1_Buffer : std_logic_vector (Data_Length downto 0) ;
    signal      Temp_2_Buffer : std_logic_vector (Data_Length downto 0) ;
    signal      Temp_3_Buffer : std_logic_vector (Data_Length downto 0) ;
    signal      Temp_4_Buffer : std_logic_vector (Data_Length downto 0) ;

    signal Fixed_Byte_Clock    : std_logic ; -- := '1' ; -- divide by 9 clock
    signal Variable_Byte_Clock : std_logic ; -- := '1' ; -- divide by 9 clock
    signal Strobe              : std_logic ; -- := '1' ; -- Used to load p-to-s register

    signal  Internal_Read_Bar : std_logic ; -- := '1' ;
    signal Internal_Write_Bar : std_logic ; -- := '1' ;
    signal   Write_A_Byte_Bar : std_logic ; -- := '1' ;
    signal      Load_Byte_Out : std_logic ; -- := '1' ;

    -- Mahyar 3/27/1996
--    signal COUNT : std_logic_vector (3 downto  0) ;

    signal Status_Reg_0  : std_logic_vector (Data_Length downto 0) ; -- := (others => '0') ;
    signal Status_Reg_1  : std_logic_vector (Data_Length downto 0) ; -- := (others => '0') ;

    signal Transfer_Bar  : std_logic ;

    signal   Sync_Detected : std_logic ;
    signal     Mode_Bit_In : std_logic ;  -- Check for '1' for command and '0' for data
    signal   Data_Mode_Bit : std_logic ;  -- set to '1' for command, '0' for data
    signal    Mode_Bit_Out : std_logic ;  -- set to '1' for command, '0' for data
    signal   Parallel_Load : std_logic ;  -- for internal use
    signal    Latched_Sync : std_logic ;

    -- These signals are drived and set based on the instructions that
    -- are part of the schedule.
    -- These signals need to be active only for one Fixed_Byte_Clock period.
    signal    Start_Transmit : std_logic ;
    signal       Start_Cycle : std_logic ;
    signal    Start_Receieve : std_logic ;
    signal     Transmit_Data : std_logic ;
    signal     Receieve_Data : std_logic ;
    signal  Data_Status_Flag : std_logic ; -- '1' for Data, '0' for Status
    signal     Start_Command : std_logic ;
```

```
    -- The following signals are used to route the FIFO data and content of
    -- status registers to the P_to_s conver module.
    -- 8/26/96
    signal       Data_Status : std_logic_vector (Data_Length downto 0) ;

    -- This signal is used to load the size of data packet into the
    -- P_to_S_Count counter.
    -- This signal needs to be active only for one Byte_Clock period.
    signal Load_P_to_S_Count : std_logic ;

    -- Need to invert the Bit_Clock
    signal Invert_Bit_Clock_In : std_logic ;

    --
    signal  Load_Command_Reg : std_logic ;
    signal Command_Data_Flag : std_logic := '0' ;

    -- Need these counters to count the number of expected data bytes in the
    -- data packets.
    signal P_to_S_Count     : std_logic_vector (Data_Length downto 0) ;
    signal S_to_P_Count     : std_logic_vector (Data_Length_Plus_1 downto 0) ;

    -- This counter is used to load in the data packet header from the
    -- Input FIFO.  It is loaded with a value of 3 and counts down to 0.
    signal Read_Data_Count  : std_logic_vector (1 downto 0) ;

    -- data packet errors while reading and writing.
    signal  Read_FIFO_Error_1 : std_logic ;
    signal  Read_FIFO_Error_2 : std_logic ;
--    signal Write_FIFO_Error_1 : std_logic ;
--    signal Write_FIFO_Error_2 : std_logic ;
    signal    Receive_Error_1 : std_logic ;
    signal    Receive_Error_2 : std_logic ;

    signal   EPROM_Error_Flag : std_logic ;

    -- Software clock to be used for checking the timing of the scheduled events.
    constant  Timer_Length : integer := 2 * Data_Length_Plus_1 ; -- 16 bits
    constant   Timer_Limit : integer := 65536 ; -- 2 ** 16
    signal           Timer : std_logic_vector ( Timer_Length - 1 downto 0 ) := (others => '0') ;
    signal     Timer_Error : std_logic ;

begin

    ------------------------------------------------------------------------
    U0:  BYTCLK port map ( Reset_BIU, Start_Cycle, Sync_Detected, Bit_Clock_In,
                           Fixed_Byte_Clock, Strobe, Variable_Byte_Clock ) ;

    U1:  S_to_P port map ( Bit_Clock_In, Serial_Data_In, Output_Data_Buffer,
                           Mode_Bit_In ) ;

    U2:  P_to_S port map ( Invert_Bit_Clock_In, Data_Status, Mode_Bit_Out, Parallel_Load,
                           Serial_Data_Out ) ;

    U3:  PSCON port map ( Load_P_to_S_Count, P_to_S_Count, Input_FIFO_Empty_Bar,
                          Bit_Clock_In, Fixed_Byte_Clock, Read_FIFO_Error_2,
                          Internal_Read_Bar ) ;

    U4:  PRMCON port map ( BIU_OR_RMU, Reset_BIU, Sync_Detected, Start_Cycle, BIU_ID,
                           Fixed_Byte_Clock, Start_Transmit, Receieve_Data, Data_Status_Flag,
                           Start_Command, MUX_Select,
                           ROM_Data, ROM_Read_Bar, ROM_Write_Bar, ROM_Address, EPROM_Error_Flag )
;

    U5: HEADER port map ( BIU_OR_RMU, Reset_BIU, BIU_ID, Mode_Bit_In, Output_Data_Buffer,
                         Variable_Byte_Clock, Sync_Detected, Command_Data_Flag,
                          Load_Command_Reg ) ;

    U6:  DATCLK port map ( Reset_BIU, Transmit_Data, Fixed_Byte_Clock, Read_Data_Count ) ;

    U7:  RECEVR port map ( Reset_BIU, Start_Cycle, Receieve_Data, Variable_Byte_Clock,
                           Load_Command_Reg, Start_Receieve, Receive_Error_1,
                           Receive_Error_2, Switch_Time_In ) ;

    U8:  STATUS port map ( BIU_OR_RMU, Reset_BIU, Start_Cycle, BIU_ID, Start_Command,
```

```
                        Start_Transmit, Data_Status_Flag, Data_Mode_Bit, FIFO_Data_In,
                        Fixed_Byte_Clock, Status_Reg_0, Transmit_Data, Load_Byte_Out,
                        Mode_Bit_Out, Data_Status ) ;

-------------------------------------------------------------------------
Byte_CLock_Out <= Fixed_Byte_Clock ;

Invert_Bit_Clock_In <= not (Bit_Clock_In) ;

Parallel_Load <= not ( Internal_Read_Bar and Load_Byte_Out ) and Strobe ;

Latched_Sync_Out <= Latched_Sync ;

Internal_Write_Bar <= ( not Variable_Byte_Clock ) or Write_A_Byte_Bar ;

-- Note: INTRPT_Out must be high for one Byte_Clock.
-- 8/28/96
INTRPT_Out <= Start_Cycle ; -- Let the uP know a new cycle started

-------------------------------------------------------------------------
-- This process stores the content of the incomming bit stream for future
-- use.  It is essential to keep this data around for a few Byte_Clock
-- cycles.  We need all the time we can get.
-- 8/2/96
-- The Temp_1_Buffer has to be loaded asynchronously to keep up with
-- possible changes and/or glitches in the incomming data bit stream.
-- 8/15/96
Load_Temp_Buffer : process ( Variable_Byte_Clock, Output_Data_Buffer,
                             Sync_Detected )

begin
  if Rising_Edge ( Sync_Detected ) then -- async load
    Temp_1_Buffer <= Output_Data_Buffer (Data_Length_Plus_1 downto 1) ;
  elsif ( Sync_Detected = '0' ) then  -- a MUST.
    if Rising_Edge ( Variable_Byte_Clock ) then
      Temp_1_Buffer <= Output_Data_Buffer (Data_Length downto 0) ;
      Temp_2_Buffer <= Temp_1_Buffer ;
      Temp_3_Buffer <= Temp_2_Buffer ;
      Temp_4_Buffer <= Temp_3_Buffer ;

    end if ;
  end if ;

end process ;

-------------------------------------------------------------------------
-- This process latches the "Sync_Detected" signal to be used by the
-- "Check_ID" state machine.  The signal is latched using the negative
-- edge of the bit clock to avoid the timing problem that otherwise
-- may occure.
--
Latch_Sync_Detected : process

begin
  wait until Falling_Edge ( Bit_Clock_In ) ;
  Latched_Sync <= Sync_Detected ;

end process ;

-------------------------------------------------------------------------
-- This process deciphers the incomming address bits and activates one of
-- the selected ports used in this module.
--
Check_More_Addresses : process ( Chip_Select_Bar, ADDRESS )

begin

  Transfer_Bar <= '1' ;

  if ( ADDRESS = PORT_ADDRESS_3 ) and ( Chip_Select_Bar = '0' ) then
    -- Transfering data from the FIFO_In to the FIFO_Out
    Transfer_Bar <= '0' ;
  end if ;

end process ;
```

```
-------------------------------------------------------------------------
Write_FIFO_Out : process -- ( Internal_Write_Bar, Input_Data_Buffer )
begin
  wait until Falling_edge ( Internal_Write_Bar ) ;
  FIFO_Data_Out <= Temp_4_Buffer ;

end process ;

-------------------------------------------------------------------------
-- The initialization of the FIFO signals are not necessary here at this
-- time.  But after implementing the global BIU reset in the hardware,
-- this process will make more sense.  For now, however, this redundancy
-- here doesn't have any sideeffects.
-- Mahyar 4/1/1996
--
Reset_FIFO_Controls: process ( Reset_BIU, Internal_Read_Bar,
                               Internal_Write_Bar )
begin
  if ( Reset_BIU = '0' ) then
    Input_FIFO_Read_Bar   <= Internal_Read_Bar ;
    Output_FIFO_Write_Bar <= Internal_Write_Bar ;
  else  -- time to reset the system.
    Input_FIFO_Read_Bar   <= '1' ;
    Output_FIFO_Write_Bar <= '1' ;
  end if ;

end process ;

-------------------------------------------------------------------------
-- The next two processes work together to define the bi-deirectional bus
--
uP_Read_XC4000_Status : process ( Data_Read_Bar, Status_Reg_0 )
begin
  if ( Data_Read_Bar = '0' ) then
    FIFO_Data_In_Out <= Status_Reg_0 ;
  else
    FIFO_Data_In_Out <= "ZZZZZZZZ" ;
  end if ;

end process ;

-------------------------------------------------------------------------
uP_Write_XC4000_Status : process -- ( Data_Write_Bar, Status_Reg_1 )
begin
  wait until Rising_edge ( Data_Write_Bar ) ;
  Status_Reg_1 <= FIFO_Data_In_Out ;

end process ;

-------------------------------------------------------------------------
-- Note: Load_P_to_S_Count need be active, i.e., high, for only a short
-- time of one Bit_Clock_Period.
-- 8/7/96
--
Read_Data_Header : process --( Fixed_Byte_Clock, Read_Data_Count, FIFO_Data_In,
                           --  Data_Mode_Bit, P_to_S_Count, Load_P_to_S_Count )
begin
  wait until Rising_Edge ( Fixed_Byte_Clock ) ; -- ==> setup time is half Fixed_Byte_Clock

  Read_FIFO_Error_1 <= '0' ;
  Data_Mode_Bit <= '0' ;  -- set to '1' for command, '0' for data
  Load_P_to_S_Count <= '0' ;

  if ( Transmit_Data = '1' ) then
    P_to_S_Count <= "00000011" ; -- read the first three header bytes.
    Load_P_to_S_Count <= '1', '0' after Bit_Clock_Period ;
    Data_Mode_Bit <= '1' ;  -- set to '1' for command, '0' for data
  else
    if ( Read_Data_Count = "11" ) then
      if ( FIFO_Data_In = "11111111" ) then  -- send out a sync-pattern
        Data_Mode_Bit <= '0' ;  -- set to '1' for command, '0' for data
      else  -- Error in data packet format
        P_to_S_Count <= "00000000" ; -- stop reading from the FIFO.
        Load_P_to_S_Count <= '1', '0' after Bit_Clock_Period ;
        Read_FIFO_Error_1 <= '1' ; -- Raise the error flag.
      end if ;
```

```
        elsif ( Read_Data_Count = "10" ) then
          if ( BIU_OR_RMU = '1' ) then -- am I BIU?
            if ( FIFO_Data_In ( Data_Length ) = '1' ) then
              -- It is a command and it is an error
              -- in data packet format and, thus, should be reported.
              P_to_S_Count <= "00000000" ; -- stop reading from the FIFO.
              Load_P_to_S_Count <= '1', '0' after Bit_Clock_Period ;
              Read_FIFO_Error_1 <= '1' ; -- Raise the error flag.
            end if ;
          end if ;

        elsif ( Read_Data_Count = "01" ) then
          P_to_S_Count  <= FIFO_Data_In ;  -- Now load the actual data count to be sent out.
          Load_P_to_S_Count <= '1', '0' after Bit_Clock_Period ;

        else -- reset all.
          Load_P_to_S_Count <= '0' ;
          P_to_S_Count <= "00000000" ; -- initialize it to zeros.

        end if ;
      end if ;

    end process ;

    ------------------------------------------------------------------------
    -- This process initiates writing of the incomming data packet to the
    -- output FIFO after detecting a synch-pattern.
    -- It keeps writing the incomming data bytes to the FIFO until the
    -- S_to_P_Count reaches zero.  The assumption is that there are as many
    -- as S_to_P_Count CONSECUTIVE data bytes comming over the serial input
    -- line.
    -- 8/1/96
    --
    Write_Out_FIFO_Process : process ( Reset_BIU, Start_Receieve, Variable_Byte_Clock,
                                       S_to_P_Count, Write_A_Byte_Bar )
      variable TEMP : integer := 0 ;

    begin
      if ( Reset_BIU = '0' ) then

        if Rising_edge ( Start_Receieve ) then
-- Synopsys vs Cadence
--          S_to_P_Count <= Temp_1_Buffer + 3 ; -- 2 to flush the Temp_i_Buffer's
          TEMP := To_integer ( Temp_1_Buffer ) ;
          TEMP := TEMP + 3 ;
          S_to_P_Count <= To_StdLogicVector ( TEMP, 9 ) ;
        end if ;

        if Falling_Edge ( Variable_Byte_Clock ) then

-- Synopsys vs Cadence
          TEMP := To_integer ( S_to_P_Count ) ;
--          TEMP := CONV_SIGNED ( S_to_P_Count ) ;

          Write_A_Byte_Bar <= '1' ;
          Start_Cycle <= '0' ;

          if ( Command_Data_Flag = '0' ) then -- it is a count of data bytes
            if ( TEMP > 0 ) then
              Write_A_Byte_Bar <= '0' ;
              TEMP := TEMP - 1 ;

-- Synopsys vs Cadence
            S_to_P_Count <= To_StdLogicVector ( TEMP, 9 ) ;
--            S_to_P_Count <= CONV_STD_LOGIC_VECTOR ( TEMP, 9 ) ;

            end if ;
          else -- it is a command!?!?!?
            -- Take the appropriate actions here and then reset the command/counter
            -- register to zero to indicate end of operation.
            Start_Cycle <= '1' ; -- restart the schedule and the BIU
            S_to_P_Count <= "000000000" ;

          end if ;
        end if ;
```

```
      else  -- time to reset the system.
        Start_Cycle <= '0' ;
        S_to_P_Count <= "000000000" ;
        Write_A_Byte_Bar <= '1' ;

      end if ;

    end process ;

    -------------------------------------------------------------------------
    -- This process stores the type of errors encountered in their designated
    -- bit positions.
    -- 8/2/96
    --
    Register_Errors : process ( Reset_BIU, Read_FIFO_Error_1, Read_FIFO_Error_2,
                                Receive_Error_1, Receive_Error_2, EPROM_Error_Flag )
    begin
      if ( Reset_BIU = '1' ) then  -- time to reset.
        Status_Reg_0 <= "00000000" ;
      else

        if ( Read_FIFO_Error_1 = '1' ) then
          Status_Reg_0 ( 0 ) <= '1' ;
        elsif ( Read_FIFO_Error_2 = '1' ) then
          Status_Reg_0 ( 1 ) <= '1' ;

        elsif ( Receive_Error_1 = '1' ) then
          Status_Reg_0 ( 2 ) <= '1' ;
        elsif ( Receive_Error_2 = '1' ) then
          Status_Reg_0 ( 3 ) <= '1' ;

        elsif ( EPROM_Error_Flag = '1' ) then
          Status_Reg_0 ( 4 ) <= '1' ;

        end if ;
      end if ;

    end process ;

    -------------------------------------------------------------------------
    -- This process handels the software timer.
    -- The Timer is reset whenever BIU is reset.
    -- The Timer is reset whenever Start_Cycle is set.
    -- 8/20/96
    --
    Timer_Controller : process ( Fixed_Byte_Clock, Reset_BIU, Start_Cycle )

      variable TEMP : integer := 0 ;

    begin
-- Synopsys vs Cadence
      TEMP := To_integer ( Timer ) ;
--      TEMP := CONV_SIGNED ( Timer ) ;

      if ( Reset_BIU = '0' ) and ( Start_Cycle = '0' ) then
        if Rising_Edge ( Fixed_Byte_Clock ) then

          Timer_Error <= '0' ;

          if ( TEMP >= Timer_Limit ) then
            -- The schedule cycle is too large for this Timer.
            Timer_Error <= '1' ;
          else
            TEMP := TEMP + 1 ;
          end if ;
        end if ;

      else  -- time to reset the system.
        Timer_Error <= '0' ;
        TEMP := 0 ;

      end if ;

-- Synopsys vs Cadence
      Timer <= To_StdLogicVector ( TEMP, Timer_Length ) ;
```

```
--      Timer <= CONV_STD_LOGIC_VECTOR ( TEMP, Timer_Length ) ;

    end process ;

    ------------------------------------------------------------------------
    ------------------------------------------------------------------------

end XC4005_Behave ;
------------------------------------------------------------------------
------------------------------------------------------------------------
------------------------------------------------------------------------
-- File Name:       S_TO_P_E.VHD
-- Host Machine:    GATEWAY 486/33 (IBM AT Clone)
-- Target Machine:  GATEWAY 486/33 (IBM AT Clone)
-- Environment :    Model Technology VHDL Simulation for Windows (Ver 4.3f)
--                  DOS Version 6.2
-- Organization:    NASA-LaRC
-- Project:         Fly By Light - Power By Wire (FBL-PBW)
-- Author:          Mahyar R. Malekpour
-- Creation Date:   05/20/96
------------------------------------------------------------------------
-- Name/Number:
--    XC4000                                (entity)
--
-- Abstract:
--   This file contains the entity declaration for the serial to parallel
--   conversion process.
--
--   SIGNAL DEFINITION :
--
-- Acronyms/Abbreviations:
--   FBL/PBW
--   BIU - Bus Interface Unit
--
-- Dependencies:
--   IEEE.STD_LOGIC_1164
--
-- Global Objects:
--
-- Exceptions:
--
-- Machine/Compiler Dependencies:
--
-- Revisions:
--
--   Modified on:  6/3/96
--   by:           Mahyar Malekpour
--
--
------------------------------------------------------------------------

library IEEE ;
use IEEE.std_logic_1164.all ;
use WORK.CNSTNT_P.all ;


entity S_TO_P is
    PORT (
           Bit_Clock_In          : in    std_logic ;
           Serial_Data_In        : in    std_logic ;
           Parallel_Data_Out     : out   std_logic_vector ((Data_Length + 1) downto 0) ;
           Mode_Bit_Out          : out   std_logic

         ) ;

end S_TO_P ;

------------------------------------------------------------------------
------------------------------------------------------------------------
------------------------------------------------------------------------
-- File Name:       S_TO_P_A.VHD
-- Host Machine:    GATEWAY 486/33 (IBM AT Clone)
-- Target Machine:  GATEWAY 486/33 (IBM AT Clone)
-- Environment :    Model Technology VHDL Simulation for Windows (Ver 4.3f)
--                  DOS Version 6.2
-- Organization:    NASA-LaRC
```

```
-- Project:         Fly By Light - Power By Wire (FBL-PBW)
-- Author:          Mahyar R. Malekpour
-- Creation Date:   05/20/96
--------------------------------------------------------------------------------
-- Name/Number:
--    S_TO_P                                (architecture)
--
-- Abstract:
--  This file contains the architecture for the serial to parallel conversion
--  process.
--
--   SIGNAL DEFINITION :
--
-- Acronyms/Abbreviations:
--   BIU - Bus Interface Unit
--
-- Dependencies:
--    IEEE.STD_LOGIC_1164
--
-- Global Objects:
--
-- Exceptions:
--
-- Machine/Compiler Dependencies:
--
-- Revisions:
--
--   Modified on:  6/3/96
--   by:           Mahyar Malekpour
--
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------

library IEEE ;
use IEEE.std_logic_1164.all ;

architecture S_TO_P_Behave of S_TO_P is

    ------------------------------------------------------------------------
    component USLR
    GENERIC ( Gen_Data_Length : Natural := Data_Length + 1 ) ;
    PORT (
           Bit_Clock_In          : in    std_logic ;

           Parallel_Data_In      : in    std_logic_vector (Gen_Data_Length downto 0) ;
           Parallel_Data_Out     : out   std_logic_vector (Gen_Data_Length downto 0) ;
           Load_Parallel         : in    std_logic ;

           Mode_Bit_In           : in    std_logic ;
           Serial_Data_In        : in    std_logic ;
           Serial_Data_Out       : out   std_logic

         ) ;

     end component ;

    ------------------------------------------------------------------------
    for all : USLR use entity work.USLR (USLR_Behave) ;

    ------------------------------------------------------------------------
    signal GND_1 : std_logic_vector (Data_Length + 1 downto 0) := "000000000" ;
    signal GND_2 : std_logic := '0' ;


begin

    ------------------------------------------------------------------------
    -- This process samples the incomming serail data bits using the falling
    -- edge of the bit clock and stores them in the Output_Data_Buffer.
    -- Note: The first bit is assumed to be the Mode_Bit and the next eight
    --       bits the data byte with the MS bit comming in first.
    --

    U0:  USLR port map ( Bit_Clock_In, GND_1, Parallel_Data_Out, GND_2, GND_2,
                         Serial_Data_In, Mode_Bit_Out ) ;
```

```
    ---------------------------------------------------------------------

end S_TO_P_Behave ;
-------------------------------------------------------------------------------
-------------------------------------------------------------------------------
-------------------------------------------------------------------------------
-- File Name:       USLR_E.VHD
-- Host Machine:    GATEWAY 486/33 (IBM AT Clone)
-- Target Machine:  GATEWAY 486/33 (IBM AT Clone)
-- Environment :    Model Technology VHDL Simulation for Windows (Ver 4.3f)
--                  DOS Version 6.2
-- Organization:    NASA-LaRC
-- Project:         Fly By Light - Power By Wire (FBL-PBW)
-- Author:          Mahyar R. Malekpour
-- Creation Date:   05/21/96
-------------------------------------------------------------------------------
-- Name/Number:
--    SHFREG                                (entity)
--
-- Abstract:
--   This file contains the entity declaration for the universal shift left
-- register with parallel in and parallel out as well as serail in and serial
-- out functionality.  However, it only shifts left one bit at a time.
--
--   conversion process.
--
--   SIGNAL DEFINITION :
--
-- Acronyms/Abbreviations:
--   FBL/PBW
--   BIU - Bus Interface Unit
--
-- Dependencies:
--   IEEE.STD_LOGIC_1164
--
-- Global Objects:
--
-- Exceptions:
--
-- Machine/Compiler Dependencies:
--
-- Revisions:
--
--   Modified on:  6/3/96
--   by:           Mahyar Malekpour
-- Introduced the generic parameter "Gen_Data_Length" to make this entitiy
-- more versetile for future use in other modules.  The default for this
-- parameter is the global constant "Data_Length".
--
-------------------------------------------------------------------------------

library IEEE ;
use IEEE.std_logic_1164.all ;
use WORK.CNSTNT_P.all ;


entity USLR is
    GENERIC ( Gen_Data_Length : Natural := Data_Length ) ;
    PORT (
           Bit_Clock_In          : in    std_logic ;

           Parallel_Data_In      : in    std_logic_vector (Gen_Data_Length downto 0) ;
           Parallel_Data_Out     : out   std_logic_vector (Gen_Data_Length downto 0) ;
           Load_Parallel         : in    std_logic ;

           Mode_Bit_In           : in    std_logic ;
           Serial_Data_In        : in    std_logic ;
           Serial_Data_Out       : out   std_logic

         ) ;

end USLR ;

-------------------------------------------------------------------------------
-------------------------------------------------------------------------------
-------------------------------------------------------------------------------
```

```
-- File Name:       USLR_A.VHD
-- Host Machine:    GATEWAY 486/33 (IBM AT Clone)
-- Target Machine:  GATEWAY 486/33 (IBM AT Clone)
-- Environment :    Model Technology VHDL Simulation for Windows (Ver 4.3f)
--                  DOS Version 6.2
-- Organization:    NASA-LaRC
-- Project:         Fly By Light - Power By Wire (FBL-PBW)
-- Author:          Mahyar R. Malekpour
-- Creation Date:   05/21/96
-------------------------------------------------------------------------------
-- Name/Number:
--    USLR                                (architecture)
--
-- Abstract:
--   This file contains the entity declaration for the universal shift left
-- register with parallel in and parallel out as well as serail in and serial
-- out functionality.  However, it only shifts left one bit at a time.
--
--   SIGNAL DEFINITION :
--
-- Acronyms/Abbreviations:
--   BIU - Bus Interface Unit
--
-- Dependencies:
--    IEEE.STD_LOGIC_1164
--
-- Global Objects:
--
-- Exceptions:
--
-- Machine/Compiler Dependencies:
--
-- Revisions:
--
--   Modified on:  6/3/96
--   by:           Mahyar Malekpour
-- Introduced the generic parameter "Gen_Data_Length" to make this entitiy
-- more versetile for future use in other modules.  The default for this
-- parameter is the global constant "Data_Length".
--
-------------------------------------------------------------------------------
-------------------------------------------------------------------------------

library IEEE ;
use IEEE.std_logic_1164.all ;
--use WORK.CNSTNT_P.all ;
--use ieee.std_logic_arith.all ;


architecture  USLR_Behave of USLR is

    signal Input_Data_Buffer  : std_logic_vector (Gen_Data_Length downto 0) ;
    signal Mode_Bit : std_logic ;


begin

    -------------------------------------------------------------------------
    Parallel_Data_Out <= Input_Data_Buffer ;
    Serial_Data_Out <= Mode_Bit ;

    -------------------------------------------------------------------------
    -- This process can load in parallel data and serializes the data byte in
    -- the Input_Data_Buffer using the rising edge of the bit clock and sends
    -- them out one bit at a time at the rising edge.
    -- This process can also load in serial data bits and send them out in
    -- parallel.
    --
    Parallel_To_Serial_To_Parallel : process
      variable Temp_Out : std_logic_vector (Gen_Data_Length downto 0) ;

    begin
      wait until Rising_Edge ( Bit_Clock_In ) ;

      if ( Load_Parallel = '1' ) then
        Input_Data_Buffer <= Parallel_Data_In ;
```

```
        Mode_Bit <= Mode_Bit_In ;

      else
        -- Convert it to serial bits and send them out,
        -- and load in serial bit.
        Temp_Out := Input_Data_Buffer ;

        Mode_Bit <= Input_Data_Buffer ( Gen_Data_Length ) ; -- MSB

        for I in Gen_Data_Length downto 1 loop
          Temp_Out ( I ) := Temp_Out ( I - 1 ) ;
        end loop ;
        Temp_Out ( 0 ) := Serial_Data_In ;

        Input_Data_Buffer <= Temp_Out  ;

      end if ;

    end process ;

    -------------------------------------------------------------------------

end USLR_Behave ;
-------------------------------------------------------------------------------
-------------------------------------------------------------------------------
-------------------------------------------------------------------------------
-- File Name:       P_TO_S_E.VHD
-- Host Machine:    GATEWAY 486/33 (IBM AT Clone)
-- Target Machine:  GATEWAY 486/33 (IBM AT Clone)
-- Environment :    Model Technology VHDL Simulation for Windows (Ver 4.3f)
--                  DOS Version 6.2
-- Organization:    NASA-LaRC
-- Project:         Fly By Light - Power By Wire (FBL-PBW)
-- Author:          Mahyar R. Malekpour
-- Creation Date:   05/20/96
-------------------------------------------------------------------------------
-- Name/Number:
--    P_TO_S                                (entity)
--
-- Abstract:
--   This file contains the entity declaration for the serial to parallel
--   conversion process.
--
--   SIGNAL DEFINITION :
--
-- Acronyms/Abbreviations:
--   FBL/PBW
--   BIU - Bus Interface Unit
--
-- Dependencies:
--   IEEE.STD_LOGIC_1164
--
-- Global Objects:
--
-- Exceptions:
--
-- Machine/Compiler Dependencies:
--
-- Revisions:
--
--   Modified on:  ??/??/96
--   by:           Mahyar Malekpour
--
--
-------------------------------------------------------------------------------

library IEEE ;
use IEEE.std_logic_1164.all ;
use WORK.CNSTNT_P.all ;


entity P_TO_S is
    PORT (
           Bit_Clock_In        : in    std_logic ;

           Parallel_Data_In    : in    std_logic_vector (Data_Length downto 0) ;
```

```
          Mode_Bit_In         : in    std_logic ;
          Load_Parallel       : in    std_logic ;

          Serial_Data_Out     : out   std_logic

        ) ;

end P_TO_S ;

-------------------------------------------------------------------------------
-------------------------------------------------------------------------------
-------------------------------------------------------------------------------
-- File Name:       P_TO_S_A.VHD
-- Host Machine:    GATEWAY 486/33 (IBM AT Clone)
-- Target Machine:  GATEWAY 486/33 (IBM AT Clone)
-- Environment :    Model Technology VHDL Simulation for Windows (Ver 4.3f)
--                  DOS Version 6.2
-- Organization:    NASA-LaRC
-- Project:         Fly By Light - Power By Wire (FBL-PBW)
-- Author:          Mahyar R. Malekpour
-- Creation Date:   05/20/96
-------------------------------------------------------------------------------
-- Name/Number:
--    P_TO_S                                (architecture)
--
-- Abstract:
--  This file contains the architecture for the parallel to serial conversion
--  process.
--
--   SIGNAL DEFINITION :
--
-- Acronyms/Abbreviations:
--   BIU - Bus Interface Unit
--
-- Dependencies:
--    IEEE.STD_LOGIC_1164
--
-- Global Objects:
--
-- Exceptions:
--
-- Machine/Compiler Dependencies:
--
-- Revisions:
--
--   Modified on:  ??/??/96
--   by:           Mahyar Malekpour
--
-------------------------------------------------------------------------------
-------------------------------------------------------------------------------

library IEEE ;
use IEEE.std_logic_1164.all ;

architecture P_TO_S_Behave of P_TO_S is

    ---------------------------------------------------------------------------
    component USLR
    PORT (
          Bit_Clock_In          : in    std_logic ;

          Parallel_Data_In      : in    std_logic_vector (Data_Length downto 0) ;
          Parallel_Data_Out     : out   std_logic_vector (Data_Length downto 0) ;
          Load_Parallel         : in    std_logic ;

          Mode_Bit_In           : in    std_logic ;
          Serial_Data_In        : in    std_logic ;
          Serial_Data_Out       : out   std_logic

        ) ;
    end component ;

    ---------------------------------------------------------------------------
    for all : USLR use entity work.USLR (USLR_Behave) ;

    ---------------------------------------------------------------------------
```

```
    signal GND_1 : std_logic_vector (Data_Length downto 0) ; -- := "00000000" ;
    signal GND_2 : std_logic := '0' ;


begin

    ------------------------------------------------------------------------
    -- This process serializes the data byte in the Input_Data_Buffer using
    -- the rising edge of the bit clock and sends them out one bit at a time
    -- at the rising edge.
    -- Note: A start data bit, i.e., '0', is send out followed by the data
    --       starting with the MS bit.
    -- This process continneously sends out a stream of bits.  When the
    -- buffer is empty, it sends out '0's.
    --
    U0:  USLR port map ( Bit_Clock_In, Parallel_Data_In, GND_1, Load_Parallel,
                         Mode_Bit_In, GND_2, Serial_Data_Out ) ;

    ------------------------------------------------------------------------

end P_TO_S_Behave ;
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
-- File Name:       PSCON_E.VHD
-- Host Machine:
-- Target Machine:
-- Environment :    Model Technology VHDL Simulation for Windows (Ver 4.3f)
--                  DOS Version 6.2
-- Organization:    NASA-LaRC
-- Project:         Fly By Light - Power By Wire (FBL-PBW)
-- Author:          Jerry H. Tucker, Mahyar Malekpour
-- Creation Date:   05/20/96
--------------------------------------------------------------------------------
-- Name/Number:
--    PSCON                                 (entity)
--
-- Abstract:
--   Synthesiable Controller for Parallel to serial convertor.
--
--   SIGNAL DEFINITION :
--
-- Acronyms/Abbreviations:
--   FBL/PBW
--   BIU - Bus Interface Unit
--
-- Dependencies:
--   IEEE.STD_LOGIC_1164
--
-- Global Objects:
--
-- Exceptions:
--
-- Machine/Compiler Dependencies:
--
-- Revisions:
--
--   Modified on:  6/12/96
--   by:           Mahyar Malekpour
-- 1.  Added this document template and,
-- 2.  changed some signal names so that they are more descriptive:
--
-- Old Name           New Name
-- ................................
-- DO                 Load_P_TO_S_Count
-- EMPTY              FIFO_Empty_Bar
-- CLK                Bit_Clock
--
--   Modified on:  6/18/96
--   by:           Mahyar Malekpour
-- Added Count_Value so that the S_TO_P_Count can be initialized to the
-- proper value.  This counter is decremented after every read from the
-- FIFO_In.
--
--   Modified on:  8/20/96
--   by:           Mahyar Malekpour
```

```
-- Added Read_FIFO_Error to report errors while reading data bytes from the
-- FIFO.
--
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------

library IEEE ;
use IEEE.std_logic_1164.all ;
use WORK.CNSTNT_P.all ;


Entity PSCON is
    port (
         Load_P_TO_S_Count : in    STD_LOGIC ;
         Count_Value       : in    std_logic_vector (Data_Length downto 0) ;
         FIFO_Empty_Bar    : in    STD_LOGIC ;
         Bit_Clock         : in    STD_LOGIC ;
         BYTE_CLOCK        : in    STD_LOGIC ;
         Read_FIFO_Error   : out   std_logic ;
         FIFO_RD_bar       : out   STD_LOGIC ) ;

end PSCON ;
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
-- File Name:       PSCON_A.VHD
-- Host Machine:    GATEWAY 486/33 (IBM AT Clone)
-- Target Machine:
-- Environment :    Model Technology VHDL Simulation for Windows (Ver 4.3f)
--                  DOS Version 6.2
-- Organization:    NASA-LaRC
-- Project:         Fly By Light - Power By Wire (FBL-PBW)
-- Author:          Jerry H. Tucker, Mahyar Malekpour
-- Creation Date:   05/20/96
--------------------------------------------------------------------------------
-- Name/Number:
--    PSCON                                 (architrecture)
--
-- Abstract:
--   Synthesiable Controller for Parallel to serial convertor.
--
--   SIGNAL DEFINITION :
--
-- Acronyms/Abbreviations:
--   FBL/PBW
--   BIU - Bus Interface Unit
--
-- Dependencies:
--   IEEE.STD_LOGIC_1164
--
-- Global Objects:
--
-- Exceptions:
--
-- Machine/Compiler Dependencies:
--
-- Revisions:
--
--   Modified on:  6/12/96
--   by:           Mahyar Malekpour
-- 1.  Added this document template and,
-- 2.  changed some signal names so that they are more descriptive:
--
-- Old Name           New Name
-- ................................
-- DO                 Load_P_TO_S_Count
-- EMPTY              FIFO_Empty_Bar
-- CLK                Bit_Clock
-- SYN                PSCON_Behave
--
-- 3. Chnaged the logic to reflect the proper logic of FIFO_Empty_Bar.
-- FIFO_Empty_Bar = '1' ==> FIFO is not empty.
--
--   Modified on:  6/18/96
--   by:           Mahyar Malekpour
-- Added a new process to handel the P_TO_S_Count counter.
```

```
--
--   Modified on:  7/31/96
--   by:           Mahyar Malekpour
-- Synch'ed the state machine with the Byte_Clock via Read_A_Byte signal.
--
--   Modified on:  8/20/96
--   by:           Mahyar Malekpour
-- Generating the FIFO read error if the FIFO is empty while reading data.
-- I raise the error flag.
--
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------

library IEEE ;
use WORK.CNSTNT_P.all ;
use ieee.std_logic_arith.all ;
use IEEE.std_logic_1164.all ;
--use ieee.std_logic_signed.CONV_INTEGER ;
use work.my_std_logic_arith.all ;


architecture PSCON_Behave of PSCON is

    signal  P_TO_S_Count : std_logic_vector (Data_Length downto 0) := (others => '0') ;
    signal   Read_A_Byte : std_logic := '1' ;

begin

    FIFO_RD_bar <= BYTE_CLOCK or Read_A_Byte ;

    --------------------------------------------------------------------------
    Read_FIFO : process ( FIFO_Empty_Bar, P_TO_S_Count )
    begin
      -- added the following statement to synch the state machine with
      -- the Byte_Clock, otherwise the FIFO_RD_bar will not be active for
      -- enough time.
      -- 7/31/96

      if (P_TO_S_Count /= "00000000") then
        if (FIFO_Empty_Bar = '1') then -- If there is data
          Read_A_Byte <= '0' after Delay_5_ns ;
          Read_FIFO_Error <= '0' after Delay_5_ns ;
        else -- FIFO is empty
          Read_A_Byte <= '1' after Delay_5_ns ;
          Read_FIFO_Error <= '1' after Delay_5_ns ;
        end if ;
      else
        Read_A_Byte <= '1' after Delay_5_ns ;
        Read_FIFO_Error <= '0' after Delay_5_ns ;
      end if ;

    end process ;

    --------------------------------------------------------------------------
    -- This process loads the counter upon Load_P_TO_S_Count active.
    -- This process decrements the counter until it reaches zero.
    --
    Counter_Controller : process ( FIFO_Empty_Bar, Load_P_TO_S_Count, Byte_Clock )
      variable  TEMP : integer ; --natural range 0 to 255 := 0 ;

    begin
      if ( Load_P_TO_S_Count = '1' ) then
        P_TO_S_Count <= Count_Value ;
      end if ;

      if Rising_Edge ( Byte_Clock ) then
-- Synopsys vs Cadence
          TEMP := To_Integer ( P_TO_S_Count ) ;
--          TEMP := CONV_INTEGER ( P_TO_S_Count ) ;

          if ( TEMP > 0 ) then
            TEMP := TEMP - 1 ;
          end if ;

-- Synopsys vs Cadence
          P_TO_S_Count <= To_StdLogicVector ( TEMP, 8 ) ;
```

```
--          P_TO_S_Count <= CONV_STD_LOGIC_VECTOR ( TEMP, 8 ) ;

      end if ;

    end process ;
    -------------------------------------------------------------------------

end PSCON_Behave;
-------------------------------------------------------------------------------
-------------------------------------------------------------------------------
-------------------------------------------------------------------------------
-- File Name:       BYTCLK_E.VHD
-- Host Machine:    GATEWAY 486/33 (IBM AT Clone)
-- Target Machine:  GATEWAY 486/33 (IBM AT Clone)
-- Environment :    Model Technology VHDL Simulation for Windows (Ver 4.3f)
--                  DOS Version 6.2
-- Organization:    NASA-LaRC
-- Project:         Fly By Light - Power By Wire (FBL-PBW)
-- Author:          Mahyar R. Malekpour
-- Creation Date:   7/26/1996
-------------------------------------------------------------------------------
-- Name/Number:
--    BYTCLK                                (architecture)
--
-- Abstract:
--   This file contains the entity for the Byte Clock generator.
--
--   SIGNAL DEFINITION :
--
-- Acronyms/Abbreviations:
--
-- Dependencies:
--    IEEE.STD_LOGIC_1164
--
-- Global Objects:
--
-- Exceptions:
--
-- Machine/Compiler Dependencies:
--
-- Revisions:
--
--   Modified on:  8/15/96
--   by:           Mahyar Malekpour
-- Separating the Byte_Clock to handel the Sync_Detected.
--
-------------------------------------------------------------------------------
-------------------------------------------------------------------------------

library IEEE ;
use ieee.std_logic_1164.all;


entity BYTCLK is
    port ( Reset_BIU               : in    std_logic ;
           Start_Cycle             : in    std_logic ;
           Sync_Detected           : in    std_logic ;
           Bit_Clock_In            : in    std_logic ;
           Fixed_Byte_Clock_Out    : out   std_logic ;
           Strobe_Out              : out   std_logic ;
           Variable_Byte_Clock_Out : out   std_logic
    ) ;

end BYTCLK ;
-------------------------------------------------------------------------------
-------------------------------------------------------------------------------
-------------------------------------------------------------------------------
-- File Name:       BYTCLK_A.VHD
-- Host Machine:    GATEWAY 486/33 (IBM AT Clone)
-- Target Machine:  GATEWAY 486/33 (IBM AT Clone)
-- Environment :    Model Technology VHDL Simulation for Windows (Ver 4.3f)
--                  DOS Version 6.2
-- Organization:    NASA-LaRC
-- Project:         Fly By Light - Power By Wire (FBL-PBW)
-- Author:          Mahyar R. Malekpour
-- Creation Date:   7/26/1996
```

```
--------------------------------------------------------------------------------
-- Name/Number:
--    BYTCLK                                (architecture)
--
-- Abstract:
--   This file contains the architecture for the Byte Clock generator.
--
--   SIGNAL DEFINITION :
--
-- Acronyms/Abbreviations:
--
-- Dependencies:
--    IEEE.STD_LOGIC_1164
--
-- Global Objects:
--
-- Exceptions:
--
-- Machine/Compiler Dependencies:
--
-- Revisions:
--
--   Modified on:  8/1/96
--   by:           Mahyar Malekpour
-- Modified the shape of the Byte_Clock and Nibble_Clock clocks while
-- maitaining the relative position of the Strobe clock.
--
--   Modified on:  8/15/96
--   by:           Mahyar Malekpour
-- Separating the Byte_Clock to handel the Sync_Detected.
--
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------

library IEEE ;
use WORK.CNSTNT_P.all ;
use ieee.std_logic_arith.all ;
use work.my_std_logic_arith.all ;


architecture BYTCLK_Behave of BYTCLK is

    ------------------------------------------------------------------------
    signal Fixed_Byte_Clock    : std_logic ; -- := '1' ; -- divide by 9 clock
    signal Variable_Byte_Clock : std_logic ; -- := '1' ; -- divide by 9 clock
    signal Nibble_Clock        : std_logic ; -- := '1' ; -- divide by 9 clock times 2, i.e.,
divide by 4 clock
    signal Strobe              : std_logic ; -- := '1' ; -- Used to load p-to-s register


begin

    Fixed_Byte_Clock_Out    <= Fixed_Byte_Clock after Delay_7_ns ;
    Variable_Byte_Clock_Out <= Variable_Byte_Clock after Delay_7_ns ;
    Strobe_Out              <= Strobe after Delay_7_ns ;

    ------------------------------------------------------------------------
    -- This process builds a 4-bit counter that counts from 0 to 8.  This counter
    -- is used to divied the incoming bit-clock by nine and assigns it to
    -- Nibble_Clock and Fixed_Byte_Clock.
    -- The counter is reset whenever BIU is reset or when a sync pattern is
    -- detected.
    --
    -- This process divides the incoming bit-clock by nine and assigns it to
    -- Fixed_Byte_Clock.  Since nine is an odd number, the Fixed_Byte_Clock
    -- will be high for 4 bit-clocks and low for 5 bit-clock.
    -- It is essential that the Fixed_Byte_Clock to be low when Count is "0000".
    --
    Fixed_Clock_Counter: process ( Reset_BIU, Bit_Clock_In, Start_Cycle )
      variable Count : natural range 0 to 15 := 0 ;

    begin

      if Rising_Edge ( Bit_Clock_In ) then

        if ( Reset_BIU = '0' ) and ( Start_Cycle = '0' ) then
```

```
          Count := ( Count + 1 ) mod 9 ;

        elsif ( Reset_BIU = '1' ) then -- Time to reset the system and intialize the counter.
          Count := 0 ;

        elsif ( Start_Cycle = '1' ) then -- Time to reset the system and intialize the counter.
          Count := 0 ;

        end if ;

        if ( Count = 2 ) then
          Nibble_Clock <= '1' ; -- not Nibble_Clock ;
        elsif ( Count = 3 ) then
          Strobe <= '1' ; -- active before the rising edge of Fixed_Byte_Clock
        elsif ( Count = 4 ) then
          Fixed_Byte_Clock   <= '1' ; -- not Fixed_Byte_Clock ;
          Nibble_Clock <= '0' ; -- not Nibble_Clock ;
          Strobe <= '0' ;
        elsif ( Count = 6 ) then
          Nibble_Clock <= '1' ; -- not Nibble_Clock ;
        elsif ( Count = 0 ) then
          Fixed_Byte_Clock   <= '0' ;
          Nibble_Clock <= '0' ;
          Strobe <= '0' ;
        end if ;

      end if ;

    end process Fixed_Clock_Counter ;

    ------------------------------------------------------------------------
    -- This process builds a 4-bit counter that counts from 0 to 8.  This
    -- counter is used to divied the incoming bit-clock by nine and assigns
    -- it to Variable_Byte_Clock.
    -- The counter is reset whenever BIU is reset or when a sync pattern is
    -- detected.
    --
    Variabl_Clock_Counter: process ( Reset_BIU, Bit_Clock_In, Sync_Detected )
      variable TEMP : natural range 0 to 15 := 0 ;

    begin
      if Rising_Edge ( Bit_Clock_In ) then
        if ( Reset_BIU = '0' ) and ( Sync_Detected = '0' ) then
          TEMP := ( TEMP + 1 ) mod 9 ;

        elsif ( Reset_BIU = '1' ) then -- Time to reset the system and intialize the counter.
          TEMP := 0 ;

        elsif ( Sync_Detected = '1' ) then -- Time to reintialize the counter.
          TEMP := 6 ;

        end if ;

        if ( TEMP = 4 ) then
          Variable_Byte_Clock   <= '1' ; -- not Variable_Byte_Clock ;
        elsif ( TEMP = 0 ) then
          Variable_Byte_Clock   <= '0' ;
        end if ;

      end if ;

    end process ;

    ------------------------------------------------------------------------

end BYTCLK_Behave ;
------------------------------------------------------------------------------
------------------------------------------------------------------------------
------------------------------------------------------------------------------
-- File Name:       HEADER_E.VHD
-- Host Machine:    GATEWAY 486/33 (IBM AT Clone)
-- Target Machine:  GATEWAY 486/33 (IBM AT Clone)
-- Environment :    Model Technology VHDL Simulation for Windows (Ver 4.4j)
--                  DOS Version 6.2
-- Organization:    NASA-LaRC
-- Project:         Fly By Light - Power By Wire (FBL-PBW)
```

```
-- Author:          Mahyar R. Malekpour
-- Creation Date:   8/1/1996
-------------------------------------------------------------------------
-- Name/Number:
--    HEADER                                (entity)
--
-- Abstract:
--  This module examines the data packet header of the incomming data and
-- detects the sync-pattern.  It then checks the data for the packet Id and
-- compares it against the local BIU_ID.  If a match is detected, LOAD_COUNTER
-- is asserted.  Error falgs are raised when necessary.
--
--   SIGNAL DEFINITION :
--
-- Acronyms/Abbreviations:
--
-- Dependencies:
--    IEEE.STD_LOGIC_1164
--
-- Global Objects:
--
-- Exceptions:
--
-- Machine/Compiler Dependencies:
--
-- Revisions:
--
--   Modified on:  8/7/96
--   by:           Mahyar Malekpour
-- Added Reset_BIU and Command_Data_Flag signals.
-- Command_Data_Flag is set high to indicate that the data packet is a command
-- and is set low to indicate that it is a count of data bytes that need to be
-- routed to the output FIFO.
--
--   Modified on:  8/9/96
--   by:           Mahyar Malekpour
--
--   Modified on:  9/4/96
--   by:           Mahyar Malekpour
-- Added BIU_OR_RMU to this module.
--
--
-------------------------------------------------------------------------------
-------------------------------------------------------------------------------

library IEEE ;
use ieee.std_logic_1164.all;
use WORK.CNSTNT_P.all ;


entity HEADER is
    port (
           BIU_OR_RMU         : in    std_logic ;
           Reset_BIU          : in    std_logic ;
           BIU_ID             : in    std_logic_vector ( Data_Length downto 0 ) ;
           Mode_Bit_In        : in    std_logic ;
           Data_In            : in    std_logic_vector ( Data_Length_Plus_1 downto 0 ) ;
           Byte_Clock_In      : in    std_logic ;
           Sync_Detected_Out  : out   std_logic ;
           Command_Data_Flag  : out   std_logic ;
           Load_Counter_Out   : out   std_logic
    ) ;

end HEADER ;
-------------------------------------------------------------------------------
-------------------------------------------------------------------------------
-------------------------------------------------------------------------------
-- File Name:       HEADER_A.VHD
-- Host Machine:    GATEWAY 486/33 (IBM AT Clone)
-- Target Machine:  GATEWAY 486/33 (IBM AT Clone)
-- Environment :    Model Technology VHDL Simulation for Windows (Ver 4.4j)
--                  DOS Version 6.2
-- Organization:    NASA-LaRC
-- Project:         Fly By Light - Power By Wire (FBL-PBW)
-- Author:          Mahyar R. Malekpour
-- Creation Date:   8/1/1996
```

```
--------------------------------------------------------------------------------
-- Name/Number:
--    HEADER                                (architecture)
--
-- Abstract:
--   This file contains the architecture for the data packet header module.
--
--   SIGNAL DEFINITION :
--
-- Acronyms/Abbreviations:
--
-- Dependencies:
--    IEEE.STD_LOGIC_1164
--
-- Global Objects:
--
-- Exceptions:
--
-- Machine/Compiler Dependencies:
--
-- Revisions:
--
--   Modified on:  8/1/96
-- Note: the Load_Counter_Out signal must be raised and used within a Bit_Clock
-- cycle.  Thus, in worst case I used a delay of half Bit_Clock cycle, i.e.,
-- Delay_5_ns ;
--
--   Modified on:  8/20/96
-- Incorporated the Global_BIU_ID in detecting packets for the modules.
--
--   Modified on:  9/4/96
--
--   Modified on:  9/16/96
-- Incorporated the status bit in detecting packets for the modules.
--
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------

library IEEE ;
use WORK.CNSTNT_P.all ;
use ieee.std_logic_arith.all ;

architecture HEADER_Behave of HEADER is

    ----------------------------------------------------------------------------
    signal   Sync_Detected : std_logic ;

begin

    Sync_Detected_Out <= Sync_Detected after Delay_2_ns ;

    ----------------------------------------------------------------------------
    -- This process checks the incomming data stream against the unique and
    -- predefined pattern of synchronization; Sync_Pattern.
    --
    Check_Sync_Pattern : process ( Reset_BIU, Data_In, Mode_Bit_In )

    begin
      Sync_Detected <= '0' ;

      if ( Reset_BIU = '0' ) then
        if ( Data_In = Sync_Pattern ) and ( Mode_Bit_In = '1' ) then
          Sync_Detected <= '1' ;
        end if ;
      end if ;

    end process ;

    ----------------------------------------------------------------------------
    -- The counter is reset whenever BIU is reset.
    -- The counter is set whenever Sync_Detected is set.
    --
    Two_Bit_Counter: process ( Reset_BIU, Sync_Detected, Byte_Clock_In,
                               Data_In, BIU_OR_RMU )
      variable TEMP : natural range 0 to 3 := 0 ;
```

```
    begin

      if ( Reset_BIU = '0' ) and ( Sync_Detected = '0' ) then

        if Rising_Edge ( Byte_Clock_In ) then

          Command_Data_Flag <= '0' ;

          if ( TEMP = 1 ) then
            -- BIU_IDs are only 5-bits long.  The higher three bits are reserved.
            if ( Data_In ( Data_Length - 4 downto 0 ) = BIU_ID ( Data_Length - 4 downto 0 )) --
is it mine?
              or ( Data_In ( Data_Length - 4 downto 0 ) = Global_BIU_ID ( Data_Length - 4 downto
0 )) -- is it everyones?
              or ( Data_In ( Data_Length - 1 ) = '1' ) -- is it status info?
              or ( BIU_OR_RMU = '0' ) then -- I am RMU
                TEMP := TEMP + 1 ;
                if ( Data_In ( Data_Length ) = '1' ) then -- it is a command
                  Command_Data_Flag <= '1' after Delay_5_ns ;
                end if ;
            else -- It is not mine, ignore it.
              TEMP := 0 ;
            end if ;

          elsif ( TEMP = 2 ) then
            Load_Counter_Out <= '1' after Delay_5_ns ;
            TEMP := 0 ;

          elsif ( TEMP = 0 ) then
            Load_Counter_Out <= '0' after Delay_5_ns ;

          end if ;
        end if ;

      elsif ( Sync_Detected = '1' ) then -- Time to reintialize the counter.
        TEMP := 1 ;
        Load_Counter_Out <= '0' after Delay_5_ns ;

      elsif ( Reset_BIU = '1' ) then -- Time to reset the system and intialize the counter.
        TEMP := 0 ;
        Load_Counter_Out <= '0' ;
        Command_Data_Flag <= '0' ;

      end if ;

    end process ;

    ---------------------------------------------------------------------------

end HEADER_Behave ;
-------------------------------------------------------------------------------
-------------------------------------------------------------------------------
-------------------------------------------------------------------------------
-- File Name:       DATCLK_E.VHD
-- Host Machine:    GATEWAY 486/33 (IBM AT Clone)
-- Target Machine:  GATEWAY 486/33 (IBM AT Clone)
-- Environment :    Model Technology VHDL Simulation for Windows (Ver 4.3f)
--                  DOS Version 6.2
-- Organization:    NASA-LaRC
-- Project:         Fly By Light - Power By Wire (FBL-PBW)
-- Author:          Mahyar R. Malekpour
-- Creation Date:   8/1/1996
-------------------------------------------------------------------------------
-- Name/Number:
--    DATCLK                                (architecture)
--
-- Abstract:
--   This file contains the entity for the read data packet clock generator.
-- This counter is used to load in the data packet header from the
-- Input FIFO.  It is loaded with a value of 3 and counts down to 0.
--
--   SIGNAL DEFINITION :
--
-- Acronyms/Abbreviations:
--
-- Dependencies:
```

```
--    IEEE.STD_LOGIC_1164
--
-- Global Objects:
--
-- Exceptions:
--
-- Machine/Compiler Dependencies:
--
-- Revisions:
--
--   Modified on:  ??/??/96
--   by:           Mahyar Malekpour
--
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------

library IEEE ;
use ieee.std_logic_1164.all;

entity DATCLK is
    port ( Reset_BIU       : in    std_logic ;
           Transmit_Data   : in    std_logic ;
           Byte_Clock_In   : in    std_logic ;
           Count_Value_Out : out   std_logic_vector ( 1 downto 0 )
    ) ;

end DATCLK ;
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
-- File Name:       DATCLK_A.VHD
-- Host Machine:    GATEWAY 486/33 (IBM AT Clone)
-- Target Machine:  GATEWAY 486/33 (IBM AT Clone)
-- Environment :    Model Technology VHDL Simulation for Windows (Ver 4.3f)
--                  DOS Version 6.2
-- Organization:    NASA-LaRC
-- Project:         Fly By Light - Power By Wire (FBL-PBW)
-- Author:          Mahyar R. Malekpour
-- Creation Date:   8/1/1996
--------------------------------------------------------------------------------
-- Name/Number:
--    DATCLK                                (architecture)
--
-- Abstract:
--   This file contains the architecture for the read data packet clock generator.
-- This counter is used to load in the data packet header from the
-- Input FIFO.  It is loaded with a value of 3 and counts down to 0.
--
-- 3 ==> read first byte, has to be FF
-- 2 ==> read second byte, is a BIU_ID
-- 1 ==> read third byte, is a command or a count of number of bytes to follow
--                        this value has to loaded into the p_to_s_count counter/register.
-- 0 ==> noop.
--
--   SIGNAL DEFINITION :
--
-- Acronyms/Abbreviations:
--
-- Dependencies:
--    IEEE.STD_LOGIC_1164
--
-- Global Objects:
--
-- Exceptions:
--
-- Machine/Compiler Dependencies:
--
-- Revisions:
--
--   Modified on:  8/1/96
--
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------

library IEEE ;
use WORK.CNSTNT_P.all ;
```

```
use ieee.std_logic_arith.all ;
use work.my_std_logic_arith.all ;


architecture DATCLK_Behave of DATCLK is

begin

    ---------------------------------------------------------------------------
    -- The counter is reset whenever BIU is reset.
    --
    Two_Bit_Counter: process ( Reset_BIU, Byte_Clock_In ) -- COUNT )
      variable TEMP : natural range 0 to 3 := 0 ;

    begin
      if ( Reset_BIU = '0' ) then

        if Rising_Edge ( Byte_Clock_In ) then
          if ( Transmit_Data = '1' ) then
            TEMP := 3 ;
          else
            if ( TEMP > 0 ) then
              TEMP := TEMP - 1 ;
            end if ;

          end if ;
        end if ;

      else  -- Time to reset the system and intialize the counter.
        TEMP := 0 ;
      end if ;

-- Synopsys vs Cadence
--      Count_Value_Out <= CONV_STD_LOGIC_VECTOR ( TEMP, 2 ) ;
      Count_Value_Out <= To_StdLogicVector ( TEMP, 2 ) after Delay_10_ns ;

    end process ;

    ---------------------------------------------------------------------------

end DATCLK_Behave ;
-------------------------------------------------------------------------------
-------------------------------------------------------------------------------
-------------------------------------------------------------------------------
-- File Name:       PRMCON_E.VHD
-- Host Machine:    GATEWAY 486/33 (IBM AT Clone)
-- Target Machine:  GATEWAY 486/33 (IBM AT Clone)
-- Environment :    Model Technology VHDL Simulation for Windows (Ver 4.3f)
--                  DOS Version 6.2
-- Organization:    NASA-LaRC
-- Project:         Fly By Light - Power By Wire (FBL-PBW)
-- Author:          Mahyar R. Malekpour
-- Creation Date:   8/14/1996
-------------------------------------------------------------------------------
-- Name/Number:
--    PRMCON                                (entity)
--
-- Abstract:
--  This module is the EPROM/RAM controller and handels fetching of
-- instructions from the EPROM/RAM.
--
--   SIGNAL DEFINITION :
--
-- Acronyms/Abbreviations:
--
-- Dependencies:
--    IEEE.STD_LOGIC_1164
--
-- Global Objects:
--
-- Exceptions:
--
-- Machine/Compiler Dependencies:
--
-- Revisions:
--
```

```
--   Modified on:  8/29/96
--   by:           Mahyar Malekpour
--  Added EPROM_Error_Flag to this entity.
--  Added BIU_OR_RMU to this entity.
--
--   Modified on:  9/4/96
--   by:           Mahyar Malekpour
--  Added MUX_Select and Start_Command to this entity.
--
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------

library IEEE ;
use ieee.std_logic_1164.all;
use WORK.CNSTNT_P.all ;
use WORK.EPROM_P.all ;

entity PRMCON is
    port (
           BIU_OR_RMU       : in    std_logic ;
           Reset_BIU        : in    std_logic ;
           Sync_Detected    : in    std_logic ;
           Start_Cycle      : in    std_logic ;
           BIU_ID           : in    std_logic_vector ( Data_Length downto 0 ) ;
           Byte_Clock_In    : in    std_logic ;
           Start_Transmit   : out   std_logic ;
           Start_Receieve   : out   std_logic ;
           Status_Data      : out   std_logic ;
           Start_Command    : out   std_logic ;
           MUX_Select       : out   std_logic_vector (Data_Length downto 0) ;

           ROM_Data         : in    std_logic_vector (ROM_WIDTH - 1 downto 0) ;
           ROM_Read_Bar     : out   std_Logic ; -- := '0' ;  -- active low
           ROM_Write_Bar    : out   std_Logic ; -- := '0' ;  -- active low
           ROM_ADDRESS      : out   std_logic_vector (ROM_ADDRESS_LINES - 1 downto 0) ;
           EPROM_Error_Flag : out   std_logic

    ) ;

end PRMCON ;
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
-- File Name:       PRMCON_A.VHD
-- Host Machine:    GATEWAY 486/33 (IBM AT Clone)
-- Target Machine:  GATEWAY 486/33 (IBM AT Clone)
-- Environment :    Model Technology VHDL Simulation for Windows (Ver 4.3f)
--                  DOS Version 6.2
-- Organization:    NASA-LaRC
-- Project:         Fly By Light - Power By Wire (FBL-PBW)
-- Author:          Mahyar R. Malekpour
-- Creation Date:   8/14/1996
--------------------------------------------------------------------------------
-- Name/Number:
--    PRMCON                                (architecture)
--
-- Abstract:
--   This file contains the architecture for the EPROM/RAM controller.
--
--   SIGNAL DEFINITION :
--
-- Acronyms/Abbreviations:
--
-- Dependencies:
--    IEEE.STD_LOGIC_1164
--
-- Global Objects:
--
-- Exceptions:
--
-- Machine/Compiler Dependencies:
--
-- Revisions:
--
--   Modified on:  8/29/96
--
```

```
--   Modified on:  9/3/96
-- Distingushing between BIU and RMU.  Added the necesarry logic.
--
--   Modified on:  9/6/96
-- To send the incomming data out on the read-bus ASAP, I had to modify the
-- code and add some new logic so that start_transmit signal is generated for
-- this case and at the appropriate time.
--
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------

library IEEE ;
use WORK.CNSTNT_P.all ;
use WORK.EPROM_P.all ;
use ieee.std_logic_arith.all ;
use work.my_std_logic_arith.all ;


architecture PRMCON_Behave of PRMCON is

    constant Delta_Time_Length : integer := Data_Length_Plus_1 ; -- 8 bits
    constant      Timer_Length : integer := (2 * Data_Length_Plus_1) ; -- 16 bits

    ------------------------------------------------------------------------
    signal      Load_Counter_Bar : std_logic := '1' ;

    signal    Instruction_Buffer : std_logic_vector ( Data_Length downto 0 ) := (others => '0') ;
    signal           READ_A_Inst : std_logic := '1' ;
    signal Internal_ROM_READ_Bar : std_logic := '1' ;
    signal           Decode_Inst : std_logic := '1' ;
    signal           Pause_Fetch : std_logic := '0' ;
    signal          Resume_Fetch : std_logic := '0' ;

begin

    ------------------------------------------------------------------------

    Internal_ROM_READ_Bar <=  BYTE_CLOCK_In or READ_A_Inst ;
    ROM_READ_Bar <= Internal_ROM_READ_Bar ;

    ------------------------------------------------------------------------
    -- This process loads the EPROM instruction into a temporary buffer for
    -- future use.
    Load_Instruction : process ( Reset_BIU, BIU_OR_RMU, Start_Cycle, ROM_Data,
                                 Internal_ROM_READ_Bar )

    begin
      if ( Reset_BIU = '0' ) and ( Start_Cycle = '0' ) then
        if ( Internal_ROM_READ_Bar = '0' ) then -- load it.
          Instruction_Buffer <= ROM_Data ( Data_Length downto 0 ) ;
        end if ;

      elsif ( Start_Cycle = '1' ) then -- Time to reset the system.
        if ( BIU_OR_RMU = '1' ) then -- if BIU
          Instruction_Buffer <= ( others => '0' ) ;
        end if ;

      else -- if ( Reset_BIU = '1' ) -- Time to reset the system.
        Instruction_Buffer <= ( others => '0' ) ;

      end if ;

    end process ;

    ------------------------------------------------------------------------
    -- The counter is reset whenever BIU is reset.
    -- The counter is set whenever Start_Cycle is set.
    --
    Decode_Instruction : process ( BIU_OR_RMU, Reset_BIU, Start_Cycle, Decode_Inst,
                                   Byte_Clock_In, Pause_Fetch, Resume_Fetch,
                                   Instruction_Buffer, Sync_Detected )

      variable  TEMP : integer ;

    begin
```

```
      if ( Reset_BIU = '0' ) and ( Start_Cycle = '0' ) then
        if Falling_Edge ( Decode_Inst ) then
          if ( Instruction_Buffer = "11111111" ) then
            Pause_Fetch <= '1' ;  -- end of schedule detected.

          elsif ( Instruction_Buffer ( 4 downto 0 ) = BIU_ID ( 4 downto 0 ) ) -- Is it mine?
          or ( Instruction_Buffer ( Data_Length - 4 downto 0 ) = Global_BIU_ID ( Data_Length - 4
downto 0 )) then -- is it everyones?
            if ( Instruction_Buffer ( 7 ) = '1' ) then
              Start_Transmit <= '1' ;
            end if ;
            if ( Instruction_Buffer ( 6 ) = '1' ) then
              Start_Receieve <= '1' ;
            end if ;
            if ( Instruction_Buffer ( 5 ) = '1' ) then
              Status_Data    <= '1' ;
            end if ;

          else -- so it is not mine, then

            -- RMU's operation is opposite of the BIU's
            -- I.e., While BIU is transmitting, RMU must be receiving data.
            -- 9/5/96
            -- RMU must transmit this data ASAP and unconditionally.
            -- 9/6/96
            if ( BIU_OR_RMU = '0' ) then -- am I RMU?
              if ( Instruction_Buffer ( 7 ) = '1' ) then
                Start_Receieve <= '1' ;
                MUX_Select <= Instruction_Buffer ;
              end if ;
            end if ;

          end if ;

          if  ( Instruction_Buffer ( 4 downto 0 ) = BIU_ID ( 4 downto 0 ) ) -- Is it mine?
          and ( Instruction_Buffer ( 7 ) = '1' )
          and ( Instruction_Buffer ( 5 ) = '1' )
          and ( BIU_OR_RMU = '0' ) then -- am I RMU?
            Start_Command <= '1' ;
          end if ;

        end if ;

        -- Special case of RMU.
        -- Send out data as soon as receiving them
        -- 9/6/96
        if ( Sync_Detected = '1' ) then
          if ( BIU_OR_RMU = '0' ) then -- am I RMU?
            -- This is the number of clock cycles that takes the data to
            -- go thru the RMU, i.e., pin to pin delay of RMU at this time.
            -- This value must be increased after introduction of voting or
            -- other operations on the incomming data.
            -- 9/6/96
            TEMP := 3 ;
          end if ;

        end if ;

        if Rising_Edge ( Byte_Clock_In ) then
          Start_Transmit <= '0' after Delay_5_ns ;
          Start_Receieve <= '0' after Delay_5_ns ;
          Status_Data    <= '0' after Delay_5_ns ;
          Start_Command  <= '0' after Delay_5_ns ;

          if ( TEMP > 0 ) then
            TEMP := TEMP - 1 ;
          end if ;

          if ( TEMP = 1 ) then
            Start_Transmit <= '1' after Delay_5_ns ;
            Status_Data    <= '1' after Delay_5_ns ;
          end if ;

        end if ;

      elsif ( Reset_BIU = '1' ) then -- Time to reset the system and intialize the counter.
```

```
        TEMP := 0 ;
        Pause_Fetch    <= '0' ;
        Start_Transmit <= '0' ;
        Start_Receieve <= '0' ;
        Status_Data    <= '0' ;
        Start_Command  <= '0' ;

      elsif ( Start_Cycle = '1' ) then -- Time to reset and intialize the counter.
        Pause_Fetch    <= '0' ;
        Start_Transmit <= '0' ;
        Start_Receieve <= '0' ;
        Status_Data    <= '0' ;
        Start_Command  <= '0' ;

      end if ;

      if ( Resume_Fetch = '1' ) then
        Pause_Fetch <= '0' ;
      end if ;

    end process ;

    ---------------------------------------------------------------------
    -- This process fetches one instruction at a time upon receiving the
    -- Start_Cycle signal.  Each instruction is 2 bytes long.  It loads the
    -- delta time of the next instruction into the Delta_Time_Clock counter
    -- and counts it down.  When it reaches zero, it then issues a read signal
    -- to the EPROM.  It stops/pauses reading from the EPROM upon Pause_Fetch
    -- and Pause_Fetch is asserted when the end-of-schedule delimiter is
    -- encountered.
    -- The counters are reset whenever BIU is reset.
    -- The counters are set whenever Start_Cycle is set.
    --
    Fetch_Instruction : process ( BIU_OR_RMU, Reset_BIU, Start_Cycle,
                                  Byte_Clock_In, Pause_Fetch, Resume_Fetch,
                                  ROM_Data, Internal_ROM_READ_Bar )

--      constant        ROM_DEPTH : integer := 2 ** ROM_DEPTH_BITS ;
      variable          Address : integer range 0 to ROM_DEPTH - 1 := 0 ;
      variable Delta_Time_Clock : integer range 0 to 256 := 0 ;

    begin

      if ( Reset_BIU = '0' ) and ( Start_Cycle = '0' ) then

        if Rising_Edge ( Byte_Clock_In ) then
          Decode_Inst <= '1' after Delay_5_ns ;
          READ_A_Inst <= '1' after Delay_5_ns ;

          if ( Pause_Fetch = '0' ) then
            Resume_Fetch <= '0' ;

            if ( Internal_ROM_READ_Bar = '0' ) then -- load it.
              Delta_Time_Clock := To_integer ( ROM_Data ( ROM_WIDTH - 1 downto 8 ) ) ;
              Delta_Time_Clock := Delta_Time_Clock + 1 ; -- to avoid lock up due to 0

            elsif ( Delta_Time_Clock > 0 ) then
              Delta_Time_Clock := Delta_Time_Clock - 1 ;

              if ( Delta_Time_Clock = 0 ) then
                -- It reached zero and time to decode the old opcode and read the next
instruction.
                Decode_Inst   <= '0' after Delay_5_ns ;
                -- Issue a read instrunction to the EPROM.
                READ_A_Inst   <= '0' after Delay_5_ns ;

-- Synopsys vs Cadence
--                ROM_ADDRESS <= Address after Delay_5_ns ;
-- 5/8/97
--                ROM_ADDRESS <= To_StdLogicVector ( Address, ROM_DEPTH_BITS ) after Delay_5_ns ;
                ROM_ADDRESS <= To_StdLogicVector ( Address, ROM_ADDRESS_LINES ) after Delay_5_ns
;

                Address := ( Address + 1 ) mod ROM_DEPTH ;
                if ( Address = 0 ) then -- I.e. we read all EPROM and didn't detect end-of-
schedule?
```

```
                EPROM_Error_Flag <= '1' ; -- There was an error
              end if ;

            end if ;
          end if ;
        else -- Pause_Fetch = '1', reset the RMU
          if ( BIU_OR_RMU = '0' ) then -- if RMU
            Address := 0 ; -- read first instruction
            Delta_Time_Clock := 1 ; -- This will force the reading of the first instruction
rightaway at the first Byte clock.
            Decode_Inst <= '1' after Delay_5_ns ;
            EPROM_Error_Flag <= '0' ;
            ROM_ADDRESS <= To_StdLogicVector ( Address, ROM_ADDRESS_LINES ) after Delay_5_ns ;
            Resume_Fetch <= '1' ;
          end if ;
        end if ;
      end if ;

    elsif ( Reset_BIU = '1' ) then -- Time to reset the system and reintialize the counter.
      if ( BIU_OR_RMU = '1' ) then -- if BIU
        Address := 2 ; -- skip first two instructions
        Delta_Time_Clock := 0 ;

      else -- elseif ( BIU_OR_RMU = '0' ) then -- if RMU
        Address := 0 ; -- read first instruction
        Delta_Time_Clock := 1 ; -- This will force the reading of the first instruction
rightaway at the first Byte clock.

      end if ;

      Decode_Inst <= '1' after Delay_5_ns ;
      READ_A_Inst <= '1' after Delay_5_ns ;
      EPROM_Error_Flag <= '0' ;
      ROM_ADDRESS <= To_StdLogicVector ( Address, ROM_ADDRESS_LINES ) after Delay_5_ns ;
      Resume_Fetch <= '0' ;

    elsif ( Start_Cycle = '1' ) then -- Time to reset and reintialize the counter.
      if ( BIU_OR_RMU = '1' ) then -- if BIU
        -- It has to BIU, so restart schedule.
        Address := 2 ; -- skip first two instructions
        Delta_Time_Clock := 1 ;
        Decode_Inst <= '1' after Delay_5_ns ;
        EPROM_Error_Flag <= '0' ;
        ROM_ADDRESS <= To_StdLogicVector ( Address, ROM_ADDRESS_LINES ) after Delay_5_ns ;

      end if ;
    end if ;

  end process ;

  -------------------------------------------------------------------------

end PRMCON_Behave ;
-------------------------------------------------------------------------------
-------------------------------------------------------------------------------
-------------------------------------------------------------------------------
-- File Name:       EPROM_P.VHD
-- Host Machine:    GATEWAY 486/33 (IBM AT Clone)
-- Target Machine:  GATEWAY 486/33 (IBM AT Clone)
-- Environment :    Model Technology VHDL Simulation for Windows (Ver 4.2e)
--                  DOS Version 6.2
-- Organization:    NASA-LaRC
-- Project:         Fly By Light - Power By Wire (FBL-PBW)
-- Author:          Mahyar R. Malekpour
-- Creation Date:   8/19/96
-------------------------------------------------------------------------------
-- Name/Number:
--    EPROM_P.VHD                                  (entity/architecture)
--
-- Abstract:
--
-- Acronyms/Abbreviations:
--   FBL/PBW
--
-- Dependencies:
--   none
```

```
--
-- Global Objects:
--
-- Exceptions:
--
-- Machine/Compiler Dependencies:
--
-- Revisions:
--   Modified on:  5/8/1997
--   by:           Mahyar Malekpour
-- 1.  Updated value of the ROM_DELAY to reflect the AMD27C256-55 chip timing,
--     i.e., Output Enable to Output Delay (Toe).
-- 2.  Updated value of the ROM_WIDTH to reflect the AMD27C256-55 width.
--
--------------------------------------------------------------------------------
library IEEE ;
use IEEE.std_logic_1164.all ;

package EPROM_P is

    constant         ROM_DELAY : time    := 35 ns ; -- Output Enable to Output Delay (Toe)
    constant         ROM_DEPTH : integer := 128 ; -- actually 32768 bytes ;
    constant         ROM_WIDTH : integer := 16 ; -- 2 eproms
    constant ROM_ADDRESS_LINES : integer := 12 ;

end EPROM_P ;

--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
-- File Name:       RECEVR_E.VHD
-- Host Machine:    GATEWAY 486/33 (IBM AT Clone)
-- Target Machine:  GATEWAY 486/33 (IBM AT Clone)
-- Environment :    Model Technology VHDL Simulation for Windows (Ver 4.3f)
--                  DOS Version 6.2
-- Organization:    NASA-LaRC
-- Project:         Fly By Light - Power By Wire (FBL-PBW)
-- Author:          Mahyar R. Malekpour
-- Creation Date:   8/20/1996
--------------------------------------------------------------------------------
-- Name/Number:
--    RECEVR                                (entity)
--
-- Abstract:
--  This module is the Receiver controller that checks for the timing of
-- receiving incomming data/command packets.
--
--   SIGNAL DEFINITION :
--
-- Acronyms/Abbreviations:
--
-- Dependencies:
--    IEEE.STD_LOGIC_1164
--
-- Global Objects:
--
-- Exceptions:
--
-- Machine/Compiler Dependencies:
--
-- Revisions:
--
--   Modified on:  ??/??/96
--   by:           Mahyar Malekpour
--
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------

library IEEE ;
use ieee.std_logic_1164.all;
use WORK.CNSTNT_P.all ;

entity RECEVR is
    port ( Reset_BIU        : in    std_logic ;
           Start_Cycle      : in    std_logic ;
           Receieve_Data    : in    std_logic ;
```

```
            Byte_Clock_In    : in    std_logic ;
            Load_Command_Reg : in    std_logic ;
            Start_Receieve   : out   std_logic ;
            Receive_Error_1  : out   std_logic ;
            Receive_Error_2  : out   std_logic ;
            Switch_Time_In   : in    std_logic_vector ( 2 downto 0 )  -- three bits for now
    ) ;

end RECEVR ;
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
-- File Name:       RECEVR_A.VHD
-- Host Machine:    GATEWAY 486/33 (IBM AT Clone)
-- Target Machine:  GATEWAY 486/33 (IBM AT Clone)
-- Environment :    Model Technology VHDL Simulation for Windows (Ver 4.3f)
--                  DOS Version 6.2
-- Organization:    NASA-LaRC
-- Project:         Fly By Light - Power By Wire (FBL-PBW)
-- Author:          Mahyar R. Malekpour
-- Creation Date:   8/20/1996
--------------------------------------------------------------------------------
-- Name/Number:
--    RECEVR                                (architecture)
--
-- Abstract:
--  This module is the Receiver controller that checks for the timing of
-- receiving incomming data/command packets.
--
--   SIGNAL DEFINITION :
--
-- Acronyms/Abbreviations:
--
-- Dependencies:
--    IEEE.STD_LOGIC_1164
--
-- Global Objects:
--
-- Exceptions:
--
-- Machine/Compiler Dependencies:
--
-- Revisions:
--
--   Modified on:  ??/??/96
--
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------

library IEEE ;
use WORK.CNSTNT_P.all ;
use ieee.std_logic_arith.all ;
use work.my_std_logic_arith.all ;


architecture RECEVR_Behave of RECEVR is

begin

    --------------------------------------------------------------------------
    -- This process checks for timing of data packet arrival within the
    -- margin of Switch_Time byte clocks.  If it doesn't receive a package
    -- for this module, it raises error flags.
    -- 8/20/96
    Check_Incoming_Data_Timning : process ( Reset_BIU, Start_Cycle, Receieve_Data,
                                            Byte_Clock_In,
                                            Load_Command_Reg )

      variable Switch_Time : integer := 0 ;
      variable   Bool_Flag : std_logic := '1' ; -- TRUE
      variable        TEMP : integer := 0 ;

    begin
      if ( Reset_BIU = '0' ) and ( Start_Cycle = '0' ) then -- time to reset.

        if Rising_Edge ( Receieve_Data ) then
```

```
          Bool_Flag := '1' ;
          TEMP := 0 ;
        end if ;

        if Rising_edge ( Byte_Clock_In ) then
          if ( Bool_Flag = '1' ) then
            TEMP := TEMP + 1 ;
          end if ;
          if ( TEMP > Switch_Time ) then
            TEMP := 0 ; -- too late for the data to arraive!
            Bool_Flag := '0' ;
            Receive_Error_1 <= '1' ; -- Timing problem
          else -- reset the error flags.
            Receive_Error_1 <= '0' ;
            Receive_Error_2 <= '0' ;
          end if ;

          Start_Receieve <= '0' after Delay_5_ns ;
        end if ;

        if Rising_Edge ( Load_Command_Reg ) then
          if ( Bool_Flag = '1' ) then
            if ( TEMP <= Switch_Time ) then
              Start_Receieve <= '1' after Delay_5_ns ;
            end if ;
            Bool_Flag := '0' ; -- time to reset the flags
            TEMP := 0 ;
          else
            Receive_Error_2 <= '1' ; -- Timing problem
          end if ;
        end if ;

      else
        Switch_Time := To_integer ( Switch_Time_In ) ;
        Receive_Error_1 <= '0' ;
        Receive_Error_2 <= '0' ;
        TEMP := 0 ;
        Bool_Flag := '0' ;
        Start_Receieve <= '0' ;

      end if ;

    end process ;

    ------------------------------------------------------------------------
    ------------------------------------------------------------------------

end RECEVR_Behave ;
------------------------------------------------------------------------------
------------------------------------------------------------------------------
------------------------------------------------------------------------------
-- File Name:       STATUS_E.VHD
-- Host Machine:    GATEWAY 486/33 (IBM AT Clone)
-- Target Machine:  GATEWAY 486/33 (IBM AT Clone)
-- Environment :    Model Technology VHDL Simulation for Windows (Ver 4.3f)
--                  DOS Version 6.2
-- Organization:    NASA-LaRC
-- Project:         Fly By Light - Power By Wire (FBL-PBW)
-- Author:          Mahyar R. Malekpour
-- Creation Date:   8/26/1996
------------------------------------------------------------------------------
-- Name/Number:
--    STATUS                                (architecture)
--
-- Abstract:
--   This file contains the entity for the send status out module.
-- The counter used is set to the number of status registers + 2 for the
-- header and ID information.  The counter counts down to 0 indicating the
-- end of operation.
--
--   SIGNAL DEFINITION :
--
-- Acronyms/Abbreviations:
--
-- Dependencies:
--    IEEE.STD_LOGIC_1164
```

```
--
-- Global Objects:
--
-- Exceptions:
--
-- Machine/Compiler Dependencies:
--
-- Revisions:
--
--   Modified on:  ??/??/96
--   by:           Mahyar Malekpour
--
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------

library IEEE ;
use WORK.CNSTNT_P.all ;
use ieee.std_logic_1164.all;


entity STATUS is
    port (
           BIU_OR_RMU       : in    std_logic ;
           Reset_BIU        : in    std_logic ;
           Start_Cycle      : in    std_logic ;
           BIU_ID           : in    std_logic_vector ( Data_Length downto 0 ) ;
           Start_Command    : in    std_logic ;
           Start_Transmit   : in    std_logic ;
           Data_Status_Flag : in    std_logic ;
           Data_Mode_Bit    : in    std_logic ;
           FIFO_Data_In     : in    std_logic_vector ( Data_Length downto 0 ) ;

           Byte_Clock_In    : in    std_logic ;
           Status_Reg_In    : in    std_logic_vector ( Data_Length downto 0 ) ;

           Transmit_Data    : out   std_logic ;
           Load_Byte_Out    : out   std_logic ;
           Mode_Bit_Out     : out   std_logic ;
           Data_Status_Out  : out   std_logic_vector ( Data_Length downto 0 )
    ) ;

end STATUS ;
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
-- File Name:       STATUS_A.VHD
-- Host Machine:    GATEWAY 486/33 (IBM AT Clone)
-- Target Machine:  GATEWAY 486/33 (IBM AT Clone)
-- Environment :    Model Technology VHDL Simulation for Windows (Ver 4.3f)
--                  DOS Version 6.2
-- Organization:    NASA-LaRC
-- Project:         Fly By Light - Power By Wire (FBL-PBW)
-- Author:          Mahyar R. Malekpour
-- Creation Date:   8/26/1996
--------------------------------------------------------------------------------
-- Name/Number:
--    STATUS                                (architecture)
--
-- Abstract:
--   This file contains the architecture for the send status out module.
-- The counter used is set to the number of status registers + 2 for the
-- header and ID information.  The counter counts down to 0 indicating the
-- end of operation.
--
--   SIGNAL DEFINITION :
--
-- Acronyms/Abbreviations:
--
-- Dependencies:
--    IEEE.STD_LOGIC_1164
--
-- Global Objects:
--
-- Exceptions:
--
-- Machine/Compiler Dependencies:
```

```
--
-- Revisions:
--
--   Modified on:  9/16/96
--   by:           Mahyar Malekpour
-- Making use of another previously unused bit, the status bit.
--
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------

library IEEE ;
use WORK.CNSTNT_P.all ;
use ieee.std_logic_arith.all ;
--use work.my_std_logic_arith.all ;

architecture STATUS_Behave of STATUS is

    signal  Status_Mode_Bit : std_logic ;  -- set to '1' for command, '0' for data
    signal      Status_Info : std_logic_vector (Data_Length downto 0) ;
    signal  Transmit_Status : std_logic ;
    signal   Write_A_Byte_1 : std_logic ; -- := '1' ;
    signal   Write_A_Byte_2 : std_logic ; -- := '1' ;

    signal Command_Mode_Bit : std_logic ;  -- set to '1' for command, '0' for data
    signal      Command_Out : std_logic_vector (Data_Length downto 0) ;
    signal Transmit_Command : std_logic ;

begin

    Load_Byte_Out <= (Write_A_Byte_1 and Write_A_Byte_2) or Byte_Clock_In after Delay_5_ns ;

    ----------------------------------------------------------------------
    -- This process is a MUX and decides to send out data or status
    -- information.
    --
    Send_Data_or_Status : process ( BIU_OR_RMU, Reset_BIU, Start_Cycle, Start_Transmit,
                                    Data_Status_Flag, Command_Mode_Bit, Command_Out,
                                    Data_Mode_Bit, FIFO_Data_In,
                                    Status_Mode_Bit, Status_Info, Start_Command )

      variable Choice : integer := 0 ;

    begin
      if ( Reset_BIU = '0' ) and ( Start_Cycle = '0' ) then
        if Rising_Edge ( Start_Transmit ) then
          if ( Data_Status_Flag = '1' ) then
            if ( Start_Command = '1' ) then -- if RMU
              Choice := 2 ; -- command
            else -- BIU
              Choice := 1 ; -- data
            end if ;
          else
            Choice := 0 ; -- status, RMU and BIU
          end if ;
        end if ;

        if ( Choice = 2 ) then
          Data_Status_Out <= Command_Out ;
          Mode_Bit_Out    <= Command_Mode_Bit ;
        elsif ( Choice = 1 ) then
          Data_Status_Out <= FIFO_Data_In ;
          Mode_Bit_Out    <= Data_Mode_Bit ;
        else -- Choice = 0
          Data_Status_Out <= Status_Info ;
          Mode_Bit_Out    <= Status_Mode_Bit ;
        end if ;

      else  -- time to reset the system.
        Choice := 1 ;

      end if ;

    end process ;

    ----------------------------------------------------------------------
    -- The Count is reset whenever BIU is reset.
```

```
-- The Count is reset whenever Start_Cycle is reset.
--
Send_Status_Out : process ( Reset_BIU, Start_Cycle, Transmit_Status,
                            Byte_Clock_In, Status_Reg_In )

  variable Count : integer := 0 ;

begin
  if ( Reset_BIU = '0' ) and ( Start_Cycle = '0' ) then
    if Falling_Edge ( Byte_Clock_In ) then
      if ( Transmit_Status = '1' ) then
        Count := 5 ; -- One extra count to be compatible with the Transmit_Data case.
      end if ;

      Status_Mode_Bit <= '0' ;
      Write_A_Byte_1 <= '0' ;

      if ( Count = 4 ) then -- send out sync-pattern first
        Status_Info <= ( others => '1' ) ;
        Status_Mode_Bit <= '1' ;

      elsif ( Count = 3 ) then -- send out my id next
        Status_Info <= BIU_ID ; --
        Status_Info ( Data_Length ) <= '0' ; -- set the command bit to data.
        Status_Info ( Data_Length - 1 ) <= '1' ; -- set the status bit.

      elsif ( Count = 2 ) then -- send out the count of data bytes to follow
        Status_Info <= "00000001" ;

      elsif ( Count = 1 ) then -- send out the status now
        Status_Info <= Status_Reg_In ;

      else -- if ( Count = 0 ) then -- stop
        Write_A_Byte_1 <= '1' ;
      end if ;

      if ( Count >= 1 ) then
        Count := Count - 1 ;
      end if ;
    end if ;

  else  -- time to reset the system.
    Count := 0 ;
    Status_Info <= ( others => '0' ) ;
    Write_A_Byte_1 <= '1' ;

  end if ;

end process ;

-------------------------------------------------------------------------
Send_Commands_Out : process ( BIU_OR_RMU, Reset_BIU, Start_Cycle,
                              Transmit_Command, Byte_Clock_In )

  variable Count : integer := 0 ;

begin
  if ( Reset_BIU = '0' ) and ( Start_Cycle = '0' ) then
    if Falling_Edge ( Byte_Clock_In ) then
      if ( Transmit_Command = '1' ) then
        Count := 4 ;
      end if ;

      Command_Mode_Bit <= '0' ;
      Write_A_Byte_2 <= '0' ;

      if ( Count = 3 ) then -- send out sync-pattern first
        Command_Out <= ( others => '1' ) ;
        Command_Mode_Bit <= '1' ;

      elsif ( Count = 2 ) then -- send start_cycle command to all BIUs by
        Command_Out <= "00011111" ; -- Global_BIU_ID ;
        Command_Out ( Data_Length ) <= '1' ; -- set the command bit.

      elsif ( Count = 1 ) then -- send out the commands
        Command_Out <= "00000001" ; -- bit zero => start_cycle, for now
```

```
          else -- if ( Count = 0 ) then -- stop
            Write_A_Byte_2 <= '1' ;
          end if ;

          if ( Count >= 1 ) then
            Count := Count - 1 ;
          end if ;
        end if ;

      else  -- time to reset the system.
        Count := 0 ;
        Command_Out <= ( others => '0' ) ;
        Write_A_Byte_2 <= '1' ;

      end if ;

    end process ;

    ---------------------------------------------------------------------------
    Set_D_S_C_Flags : process ( BIU_OR_RMU, Start_Transmit, Data_Status_Flag,
                                Start_Command )

    begin
      if ( BIU_OR_RMU = '0' ) and ( Start_Command = '1' ) then -- if I am RMU
        Transmit_Command <= '1' ;
        Transmit_Data    <= '0' ;
      else  -- if I am BIU
        Transmit_Command <= '0' ;
        Transmit_Data    <= Start_Transmit and Data_Status_Flag ; -- 1 and 1
      end if ;

      Transmit_Status <= Start_Transmit and ( not Data_Status_Flag ) ; -- 1 and 0

    end process ;

    ---------------------------------------------------------------------------
    ---------------------------------------------------------------------------

end STATUS_Behave ;
-------------------------------------------------------------------------------
-------------------------------------------------------------------------------
```

Appendix B
C Codes

```
//      File:                   TESTPAL.CPP
//      Author:                 Mahyar Malekpour, Gabriel Velasquez
//      Comments:               File Documentation
//      Creation Date: November 29, 1995
//      Last Mod:               November 30, 1995
//      Comment:                Changed some variable's names

#include <stdio.h>
#include <iostream.h>
#include <string.h>
#include <stdlib.h>

void main()
{
    char one_byte ;
    int i ;

    cout << "Ready to test the tri-state signal." << endl;
    cout << "Enter a charactor to continue." << endl;
    cin >> one_byte ;

        _asm                            //inline assembly
        {
        mov dx, 0300h           //Reset and D/P_Bar Port
        mov al,01h                      //Reset High, D/P_Bar Tristate
        out dx,al
        }

        for(i=0;i<2;i++)                //Creates a delay(between 6 and 7 microsecs)
        {
        _asm                            //inline assembly
        {
               mov dx, 0300h    //Reset and D/P_Bar Port
               mov al,00h               //Reset Low, D/P_Bar Tristate
               out dx,al
        }
        }

        for(i=0;i<2;i++)                //Creates a delay(between 6 and 7 microsecs)
        {
               _asm                     //inline assembly
               {
               mov dx, 0300h    //Reset and D/P_Bar Port
               mov al,02h               //Reset Low, D/P_Bar Low
               out dx,al
               }
        }

        for(i=0;i<2;i++)
        {
               _asm
               {
               mov dx,0300h             //Reset and D/P_Bar Port
               mov al,03h               //Reset High, D/P_Bar Low
               out dx,al
               }
        }

        _asm
        {
        mov dx,0300h                    //Reset and D/P_Bar Port
        mov al,01h                      //Reset High, D/P_Bar Tristate
        out dx,al
        }

    cout << endl;
    cout << "Finished test." << endl;

} // end main
```

```
////////////////////////////////////////////////////////////////////////////
//
//      File:           XC3020.CPP
//
//      Author:         Mahyar Malekpour
//
//      Comment:
//
//      Creation Date: 4/12/96
//
//      Last Mod:       3/12/96
//    Changed the code to reflect the changes in the base addresses used
//  in the PAL and in the XC3020.
//
//  Old Address            New Address     Device    Function
//  ---------------------------------------------------------------
//  300H             306H            PAL             Reset XC3020
//  301H             307H            PAL             Program XC3020
//  302H             302H            XC3020          Write status (Reset FIFOs)
//  303H             300H            XC3020          Read/Write FIFOs
//  ----                301H            XC3020       Read status of FIFOs
//  ----                303H            XC4000       Transfer Data
//  ----                304H            reserved     reserved
//  ----                305H            XC3020       Reset and Program XC4000
//
////////////////////////////////////////////////////////////////////////////


#include <stdio.h>
#include <iostream.h>
#include <string.h>
#include <stdlib.h>

void Reset_Xilinx_3000 () ;
//void WriteToXC3000 () ;

//#define ARY_SIZE 100

void main()
{
    FILE *infile_ptr ; // , *outfile_ptr ;

    int array_count = 0 ;
    int infile_len ;
    char infilename [80] ;
    int char_in ;
    char *f_ptr ;

    Reset_Xilinx_3000 () ;

    //Get the input file name from user.
    cout << "Enter Xilinx (XC3000 Family) Bitstream File Name (*.rbt): " ;
    cin >> infilename ;
    cout << endl ;

    infile_len = strlen ( infilename ) ;
    f_ptr = &infilename [ infile_len - 4 ] ;

    if ( _strnicmp ( f_ptr, ".rbt", 4 ))
    {
      cout << "Error: No rbt extention !!!!" << endl ;
      exit ( 1 ) ;
    }

    //Open input file
    if ( (infile_ptr = fopen (infilename, "r") ) == NULL)
    {
      cout << "Cannot open input file: " << infilename << endl ;
      exit ( 0 ) ;
    }
    cout << "Opened the input file: " << infilename << " to read." << endl ;

    array_count = 0 ;
    // Read the input rbt text file and save it in an array of char's.
    char_in = fgetc (infile_ptr) ;
    while ( feof (infile_ptr) == 0 )
```

```
    {
        array_count++ ;

        if ( (char) char_in == '1' )
        {
               _asm
               {
               mov dx,0307h
               mov al,01h
               out dx,al                                    //Data High
               }

        }
        else
         if ( (char) char_in == '0' )
        {
               _asm
               {
               mov dx,0307h
               mov al,00h
               out dx,al                                    //Data Low
               }

        }  // if

      // Read the next character.
      char_in = fgetc (infile_ptr) ;
    } // while

    fclose ( infile_ptr ) ;

    cout << endl ;
    cout << endl ;
    cout << "Finished reading the XC3000 rbt file." << endl;
    cout << endl << endl;
    cout << "Number of characters read:  " << array_count << endl;

} // end main

//////////////////////////////////////////////////////////////////////////////
//
// 1.  A '1' sets the D/P_Bar Low, while a '0' sets the D/P_Bar into
//     a state of high impedence 'Z'.
// 2.  Reset (bit D0) and D/P_Bar (bit D1) are at Address 306.
//     The Data Port is at Address 301.  Bit D0 is used serially.
//
//////////////////////////////////////////////////////////////////////////////

void Reset_Xilinx_3000 ()
{
        int i;

        for(i=0;i<4;i++)                    //Creates a delay(between 6 and 7 microsecs)
        {
               _asm                         //inline assembly
               {
               mov dx, 0306h                //D/P_Bar and Reset Port
               mov al,00h                   //D/P_Bar Tristate, Reset Low
               out dx,al
               }
        }

        for(i=0;i<4;i++)                    //Creates a delay(between 6 and 7 microsecs)
        {
               _asm                         //inline assembly
               {
               mov dx, 0306h                //D/P_Bar and Reset Port
               mov al,02h                   //D/P_Bar Low, Reset Low
               out dx,al
               }
        }

        for(i=0;i<4;i++)
        {
               _asm
               {
```

```
                mov dx, 0306h                   //D/P_Bar and Reset Port
                mov al,03h                      //D/P_Bar Low, Reset High
                out dx,al
                }
        }

        _asm
        {
        mov dx, 0306h                           //D/P_Bar and Reset Port
        mov al,01h                              //D/P_Bar Tristate, Reset High
        out dx,al
        }

} // Reset_Xilinx_3000

////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////
//
//      File:           XC4000.CPP
//
//      Author:         Mahyar Malekpour
//
//      Comment:
//
//      Creation Date: 4/12/96
//
//      Last Mod:       4/29/96
//    Changed the code to reflect the changes in the base addresses used
//  in the PAL and in the XC3020.
//
//  Old Address             New Address     Device    Function
//  ------------------------------------------------------------
//  300H            306H            PAL             Reset XC3020
//  301H            307H            PAL             Program XC3020
//  302H            302H            XC3020          Write status (Reset FIFOs)
//  303H            300H            XC3020          Read/Write FIFOs
//  ----               301H            XC3020       Read status of FIFOs
//  ----               303H            XC4000       Transfer Data
//  ----               304H            reserved     reserved
//  ----               305H            XC3020       Reset and Program XC4000
//
////////////////////////////////////////////////////////////////////////////

#include <stdio.h>
#include <iostream.h>
#include <string.h>
#include <stdlib.h>

void Reset_XC4000 () ;
int  XC4000_Ready () ;
int  Write_XC4000 () ;

void main()
{
    char ch = 'a' ;

    while (1)
    {
         if (( ch == 'x' ) || ( ch == 'X' ))
         {
     cout << endl << "Exiting program." << endl ;
            break ;
         }
         else if (( ch == 'r' ) || ( ch == 'R' ))
         {
           cout << "Reseting the XC4000" << endl ;
           Reset_XC4000 () ;
         }
         else if (( ch == 's' ) || ( ch == 'S' ))
         {
           cout << "Cheking status of the XC4000" << endl ;
           XC4000_Ready () ;
         }
         else if (( ch == 'p' ) || ( ch == 'P' ))
         {
```

```
            cout << "Programming the XC4000" << endl ;
            Write_XC4000 () ;
          }

        cout << endl ;
        cout << "-------------------------------" << endl ;
        cout << "Waiting for command:" << endl ;
        cout << "     r to Reset the XC4000," << endl ;
        cout << "     p to Program the XC4000," << endl ;
        cout << "     s to read Status of the XC4000, or" << endl ;
        cout << "     x to eXit this program." << endl ;
        cin >> ch ;
        cout << endl ;

    } // end while

} // end main

////////////////////////////////////////////////////////////////////////////
//
// 1.  Reset and programming of the XC4000 is done thru the same port 305Hex.
// 2.  While accessing this port, uP data bus is used to write and read to
//      to this port.
// 3.  Functions         XC4000 Pins       uP Data Bus Pins
//      ---------------------------------------------------
//      Reset                 PROG_Bar          D0
//      Program            Din                    D1
//      Read Staus        INIT                  D0
//      Wait                  DONE                D1
// 4.  Thus, while programming XC4000, D0 must be held high.
//
////////////////////////////////////////////////////////////////////////////

void Reset_XC4000 ()
{
  int i ;

        for(i=0;i<4;i++)                    // Creates a delay (between 6 and 7 microsecs)
        {
                _asm                                            // inline assembly
                {
                        mov dx, 0305h
                        mov al,00h                          // PROG_Bar Low
                        out dx,al
                }
        }

        for(i=0;i<4;i++)                    // Creates a delay (between 6 and 7 microsecs)
        {
                _asm                                            //inline assembly
                {
                        mov dx, 0305h
                        mov al,01h                          // PROG_Bar High
                        out dx,al
                }
        }

} // Reset_XC4000

////////////////////////////////////////////////////////////////////////////
int XC4000_Ready ()
{
  int INIT_In ;

        _asm                                            // inline assembly
        {
                mov dx, 0305h
        mov ah, 00h
                in  al, dx
                and al, 003h   // mask off unused bits
                mov INIT_In, ax
        }

        if ( INIT_In == 0 )
        {
        cout << "INIT is Low" << endl ;
```

```
        cout << "DONE is Low" << endl ;
        cout << "There was an error!" << endl ;
        return ( 1 ) ;
        }
        else if ( INIT_In == 1 )
        {
               cout << "INIT is High" << endl ;
        cout << "DONE is Low" << endl ;
        cout << "XC4000 is ready to be programmed." << endl ;
        }
        else if ( INIT_In == 2 )
        {
        cout << "INIT is Low" << endl ;
               cout << "DONE is High" << endl ;
        cout << "There was an error!" << endl ;
        return ( 1 ) ;
        }
        else if ( INIT_In == 3 )
        {
        cout << "INIT is High" << endl ;
        cout << "DONE is High" << endl ;
        cout << "XC4000 is ready for normal operation and if desired to be reprogrammed." << endl
;
        }
        else // This should never happen!
        {
        cout << "There was an error!" << endl ;
               cout << "XC4000 status is : " << INIT_In << endl ;
        return ( 1 ) ;
        }

        return ( 0 ) ;

} // XC4000_Ready

////////////////////////////////////////////////////////////////////////////
int Write_XC4000 ()
{
    FILE *infile_ptr ;

    int Bit_Count = 0 ;
    int infile_len ;
    char infilename [80] ;
    int char_in ;
    char *f_ptr ;
    int Frame_Error, Frame_Num ;
    int Bits_In_Frame ;  // used to prevent sending CRC bits to the xc4000.

    //Get the input file name from user.
    cout << "Enter Xilinx (XC4000 Family) Bitstream File Name (*.rbt): " ;
    cin >> infilename ;
    cout << endl ;

    infile_len = strlen ( infilename ) ;
    f_ptr = &infilename [ infile_len - 4 ] ;

    if ( _strnicmp ( f_ptr, ".rbt", 4 ))
    {
      cout << "Error: No rbt extention !!!!" << endl ;
      return ( 1 ) ;
    }

    //Open input file
    if ( (infile_ptr = fopen (infilename, "r") ) == NULL)
    {
      cout << "Cannot open input file: " << infilename << endl ;
      return ( 0 ) ;
    }
    cout << "Opened the input file: " << infilename << " to read." << endl ;

    // See notes by the Reset_XC4000 function.
    Bit_Count = 0 ;
    Frame_Num = 0 ;
    Frame_Error = 0 ;
    Bits_In_Frame = 0 ;
    // Read the input rbt text file and save it in an array of char's.
```

```
    char_in = fgetc (infile_ptr) ;
    while ( feof (infile_ptr) == 0 )
    {
        Bit_Count++ ;
        Bits_In_Frame++ ;

        if ( char_in == 10 ) // if LF
        {
          Frame_Error = XC4000_Ready () ;
          if ( Frame_Error ) // exit the function
          {
            fclose ( infile_ptr ) ;
            cout << endl ;
            cout << "There was a FRAME ERROR at the frame number " << Frame_Num << endl ;
            cout << endl ;
            cout << "Terminated reading the XC4000 rbt file." << endl ;
            cout << endl ;
            cout << "Number of characters read:  " << Bit_Count << endl ;
            cout << "Total of " << Frame_Num << " frames were written to the XC4000." << endl ;
            return ( 1 ) ;
          }

          Frame_Num++ ;
          Bits_In_Frame = 0 ;

        }
        else
        if ( (char) char_in == '1' )
        {
        _asm
        {
               mov dx,0305h
               mov al,03h

               out dx,al                                        //Data High
        }

        }
        else
        if ( (char) char_in == '0' )
        {
        _asm
        {
               mov dx,0305h
               mov al,01h

               out dx,al                                        //Data Low
        }

        }  // if

      // Read the next character.
      char_in = fgetc (infile_ptr) ;
       }  // while

    fclose ( infile_ptr ) ;

    // Check the status once more
    Frame_Error = XC4000_Ready () ;

    cout << endl ;
    cout << endl ;
    cout << "Finished reading the XC4000 rbt file." << endl ;
    cout << endl << endl;
    cout << "Number of characters read:  " << Bit_Count << endl ;
    cout << "Total of " << Frame_Num << " frames were written to the XC4000." << endl ;

    return ( 0 ) ;

} // Write_XC4000

////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////
```

Appendix C
Pin Assignments and Layouts

Following is the content of file "pal22v.npi" that describes the PALL22V10 pin assignments:

```
{XOR_POLARITY_CONTROL FALSE, MAX_XOR_PTERMS 0, POLARITY_CONTROL TRUE, MAX_PTERMS 16, MAX_SYMBOLS
20};

DEVICE
       {XOR_POLARITY_CONTROL FALSE, MAX_XOR_PTERMS 0, POLARITY_CONTROL TRUE, MAX_PTERMS 16,
MAX_SYMBOLS 20};
       TARGET 'PART_NUMBER AMD PALLV22V10-10PC';

       INPUT CLK_IN:1;
       INPUT D1:2;
       INPUT AEN:3;
       INPUT ADDRESS_0_:4;
       INPUT ADDRESS_1_:5;
       INPUT ADDRESS_2_:6;
       INPUT ADDRESS_3_:7;
       INPUT ADDRESS_4_:8;
       INPUT ADDRESS_5_:9;
       INPUT ADDRESS_6_:10;
       INPUT ADDRESS_7_:11;
       INPUT ADDRESS_8_:13;
       INPUT ADDRESS_9_:14;
       DATA_OUT:15 {XOR_POLARITY_CONTROL FALSE, MAX_XOR_PTERMS 0, POLARITY_CONTROL TRUE,
MAX_PTERMS 16, MAX_SYMBOLS 20};
       CLK_OUT:16 {XOR_POLARITY_CONTROL FALSE, MAX_XOR_PTERMS 0, POLARITY_CONTROL TRUE,
MAX_PTERMS 16, MAX_SYMBOLS 20};
       INPUT FEEDBACK_DONE_PROG:17;
       DONE_PROG_BAR:18 {XOR_POLARITY_CONTROL FALSE, MAX_XOR_PTERMS 0, POLARITY_CONTROL TRUE,
MAX_PTERMS 16, MAX_SYMBOLS 20};
       RESET_OUT:19 {XOR_POLARITY_CONTROL FALSE, MAX_XOR_PTERMS 0, POLARITY_CONTROL TRUE,
MAX_PTERMS 16, MAX_SYMBOLS 20};
       DONE_PROG_TRISTATE:20 {XOR_POLARITY_CONTROL FALSE, MAX_XOR_PTERMS 0, POLARITY_CONTROL
TRUE, MAX_PTERMS 16, MAX_SYMBOLS 20};
       INPUT IOWR_BAR:21;
       INPUT D0:22;
       X_CLK_OUT:23 {XOR_POLARITY_CONTROL FALSE, MAX_XOR_PTERMS 0, POLARITY_CONTROL TRUE,
MAX_PTERMS 16, MAX_SYMBOLS 20};

       NO_CONNECT 12, 24;
END DEVICE;

VIRTUAL DFF.mod001111.x, DFF.mod001105.x, BUFTH.mod000054.x, Xdefault_0,
           w000622, w000623, w000517, w000428,
           w000511, w000618, w000557, BUFTH.mod000054.i,
           BUFTH.mod000054.oe, BUFTH.mod000054.RETURN, fGND.mod000962.RETURN, DFF.mod001111.q,
           DFF.mod001111.q_bar, DFF.mod001111.d, DFF.mod001111.clk, DFF.mod001105.q,
           DFF.mod001105.q_bar, DFF.mod001105.d, DFF.mod001105.clk;
```

Following is the content of file "xc3020.cst" that describes the XC3020 pin assignments:

```
; Last update: Mahyar 3/28/1996
; Added Chip_Select_Bar
; Last Modified on 4/10/1996
;
; Last Modified on 5/2/1996
; DIN_4000 is tied to D(0) and PROG_4000 is tied to D(5)
; So, they don't have special pins anymore.
;
; Last Modified on 6/7/1996
; Added DIRCTION pin 44
;
place block CCLK_4000 P75;
;place block DIN_4000 P76;
place block PROG_4000 P77;
place block INIT_4000 P78;
place block DONE_4000 P66;
;
place block CHIP_SELECT_BAR P11;
place block RESET_BIU P30;
place block DIRECTION P44;
;place block UP_DATA_PIN_5 P61;
place block DATA_READ_BAR P10;
place block DATA_WRITE_BAR P25;
;
place block BIU_FIFO_WRITE_BAR P68;
place block BIU_FIFO_READ_BAR P70;
;
place block OUTPUT_FIFO_HF_BAR P52;
place block OUTPUT_FIFO_EMPTY_BAR P45;
place block OUTPUT_FIFO_FULL_BAR P46;
place block OUTPUT_FIFO_WRITE_BAR P47;
place block OUTPUT_FIFO_READ_BAR P48;
place block OUTPUT_FIFO_RESET_BAR P49;
;
place block INPUT_FIFO_HF_BAR P27;
place block INPUT_FIFO_EMPTY_BAR P29;
place block INPUT_FIFO_FULL_BAR P35;
place block INPUT_FIFO_WRITE_BAR P37;
place block INPUT_FIFO_READ_BAR P39;
place block INPUT_FIFO_RESET_BAR P40;
;
place block FIFO_DATA_IN_OUT<7> P67;
place block FIFO_DATA_IN_OUT<6> P65;
place block FIFO_DATA_IN_OUT<5> P63;
place block FIFO_DATA_IN_OUT<4> P62;
place block FIFO_DATA_IN_OUT<3> P60;
place block FIFO_DATA_IN_OUT<2> P59;
place block FIFO_DATA_IN_OUT<1> P58;
place block FIFO_DATA_IN_OUT<0> P56;
;
place block UP_DATA_IN_OUT<7> P15;
place block UP_DATA_IN_OUT<6> P16;
place block UP_DATA_IN_OUT<5> P17;
place block UP_DATA_IN_OUT<4> P18;
place block UP_DATA_IN_OUT<3> P19;
place block UP_DATA_IN_OUT<2> P20;
place block UP_DATA_IN_OUT<1> P21;
place block UP_DATA_IN_OUT<0> P23;
;
place block IORD_BAR P24;
place block IOWR_BAR P26;
place block AEN_BAR P28;
;
place block ADDRESS<9> P81;
place block ADDRESS<8> P82;
place block ADDRESS<7> P83;
place block ADDRESS<6> P84;
place block ADDRESS<5> P2;
place block ADDRESS<4> P3;
place block ADDRESS<3> P4;
place block ADDRESS<2> P5;
place block ADDRESS<1> P8;
place block ADDRESS<0> P9;
;
```

```
;place block DONE_PROG_BAR P55;
;place block SERIAL_PROG_IN P72;
;place block CLK_IN P74;
;place block RESET P54;
```

Following is the content of file "xc4000.cst" that describes the XC4000 pin assignments:

```
# Pin assignments for the XC4000
# Mahyar 3/25/1996
# Changed the format on 4/5/1996
# Last Modified on 4/12/1996
#
# Last Modified on 5/14/1996
#
place instance CHIP_SELECT_BAR : A1;
place instance RESET_BIU : A2;
#
#place instance FIFO_DATA_IN<7> : L1;
#place instance FIFO_DATA_IN<6> : L2;
#place instance FIFO_DATA_IN<5> : K1;
#place instance FIFO_DATA_IN<4> : K2;
#place instance FIFO_DATA_IN<3> : K3;
#place instance FIFO_DATA_IN<2> : J1;
#place instance FIFO_DATA_IN<1> : J2;
#place instance FIFO_DATA_IN<0> : J3;
#
place instance FIFO_DATA_OUT<7> : G1;
place instance FIFO_DATA_OUT<6> : G2;
place instance FIFO_DATA_OUT<5> : G3;
place instance FIFO_DATA_OUT<4> : F1;
place instance FIFO_DATA_OUT<3> : F2;
place instance FIFO_DATA_OUT<2> : E1;
place instance FIFO_DATA_OUT<1> : E2;
place instance FIFO_DATA_OUT<0> : E3;
#
place instance INPUT_FIFO_READ_BAR : D3;
#place instance INPUT_FIFO_HF_BAR : R11;
#place instance INPUT_FIFO_EMPTY_BAR : R10;
#place instance INPUT_FIFO_FULL_BAR : P9;
#
place instance OUTPUT_FIFO_WRITE_BAR : M3;
place instance OUTPUT_FIFO_HF_BAR : N15;
#place instance OUTPUT_FIFO_EMPTY_BAR : N14;
#place instance OUTPUT_FIFO_FULL_BAR : M14;
#
#
place instance BIT_CLOCK_IN : B3;
place instance BYTE_CLOCK_OUT : B1;
#
place instance ADDRESS<2> : T13;
place instance ADDRESS<1> : R13;
place instance ADDRESS<0> : P12;
#
place instance IORD_BAR : N1;
#place instance IOWR_BAR : C1;
#
place instance FIFO_DATA_IN_OUT<7> : T16;
place instance FIFO_DATA_IN_OUT<6> : T14;
place instance FIFO_DATA_IN_OUT<5> : T10;
place instance FIFO_DATA_IN_OUT<4> : R9;
place instance FIFO_DATA_IN_OUT<3> : T8;
place instance FIFO_DATA_IN_OUT<2> : P7;
place instance FIFO_DATA_IN_OUT<1> : T3;
place instance FIFO_DATA_IN_OUT<0> : P4;
#
place instance DATA_READ_BAR : T9;
place instance DATA_WRITE_BAR : T11;
#
#place instance DONE : R15;
#place instance SERIAL_PROG_IN : R14;
#place instance CCLK : R2;
#
#place instance BIU_DATA_IN<7> : M16;
#place instance BIU_DATA_IN<6> : L16;
```

```
#place instance BIU_DATA_IN<5> : K16;
#place instance BIU_DATA_IN<4> : J16;
#place instance BIU_DATA_IN<3> : H16;
#Place instance BIU_DATA_IN<2> : G16;
#place instance BIU_DATA_IN<1> : F16;
#place instance BIU_DATA_IN<0> : E16;
#
#place instance EPROM_DATA<7> : C5;
#place instance EPROM_DATA<6> : C7;
#place instance EPROM_DATA<5> : B5;
#place instance EPROM_DATA<4> : B6;
#place instance EPROM_DATA<3> : B7;
#Place instance EPROM_DATA<2> : A6;
#place instance EPROM_DATA<1> : A7;
#place instance EPROM_DATA<0> : A8;
#
#place instance EPROM_ADDRESS<11> : C9;
#place instance EPROM_ADDRESS<10> : C10;
#place instance EPROM_ADDRESS<9> : C12;
#place instance EPROM_ADDRESS<8> : B9;
#place instance EPROM_ADDRESS<7> : B10;
#place instance EPROM_ADDRESS<6> : B11;
#place instance EPROM_ADDRESS<5> : B12;
#place instance EPROM_ADDRESS<4> : B13;
#place instance EPROM_ADDRESS<3> : A9;
#Place instance EPROM_ADDRESS<2> : A10;
#place instance EPROM_ADDRESS<1> : A11;
#place instance EPROM_ADDRESS<0> : A13;
#
#place instance EPROM_READ : A14;
#place instance EPROM_WRITE : C15;
#
#place instance SERIAL_DATA_IN : C2;
#place instance SERIAL_DATA_OUT : N2;
#
```

Appendix D
Schedule and Data Packet Examples

Abbreviations used in the following schedule tables:

DT = Delta Time
Tx = Transmit
Rx = Receive
S/D/C = Status or Data or Command
Id = BIU or RMU Id

Schedule for the ideal test case (Content of "ideal-s.txt" file):

DT	Tx	Rx	S/D/C	Id
10	1	0	1	27
5	0	0	0	27
5	1	0	0	27
1	0	1	1	31
6	1	0	0	1
4	0	1	1	31
6	1	0	0	2
4	0	1	1	31
30	1	0	1	1
1	0	1	1	31
5	1	0	1	2
1	0	1	1	3
19	1	0	1	3
1	0	1	1	4
7	1	0	1	4
1	0	1	1	1
2	1	0	1	1
1	0	1	1	2
5	1	0	1	2
1	0	1	1	3
19	1	0	1	1
1	0	1	1	2
5	1	0	1	3
1	0	1	1	4
15	1	1	1	31

Schedule for the fail BIU test case (Content of "fail-biu.txt" file):

DT	Tx	Rx	S/D/C	Id
10	1	0	1	27
5	0	0	0	27
5	1	0	0	27
1	0	1	1	31
4	1	0	0	1
2	0	1	1	31
4	1	0	0	2
2	0	1	1	31
30	1	0	1	1
1	0	1	1	31
5	1	0	1	2
1	0	1	1	3
19	1	0	1	3
1	0	1	1	4
7	1	0	1	4
1	0	1	1	1
2	1	0	1	1
1	0	1	1	2
5	1	0	1	2
1	0	1	1	3
19	1	0	1	1
1	0	1	1	2
5	1	0	1	3
1	0	1	1	4
15	1	1	1	31

Appendix E
VHDL Tools

This appendix contains the design flow and procedures necessary to get through the many tools of VHDL development environment. Specifically, the Cadence tool set consisting of "picdesign", "picxilinx", "hdldesk", and "Synergy".

This procedure is put together to help new users to get thru Cadence tools and to speed up the initial learning curve.

Last Modified on: 11-21-95
Cadence Version: 9404

Procedure for using the many files and tools of Cadence development environment to synthesize and implement a given design in VHDL:

1. run "hdldesk &"
2. compile all codes using hdldesk
3. run "SynrgCheck" from the hdldesk
4. "Synthesize" from hdldesk
5. select the target library; 3000, 4000, or etc.
6. set synthesis option to cost (a must for XC3000),
   "yes" for schematic generation, and
   select "STD Logic" option
7. Synergy brings up a pop-up window titled "Import VHDL",
   select "Verilog Model Import Files" and OK it.
   Check for warnings and errors in the log files.
8. quit Synergy
9. edit "xnf_out" file and specify the latest run directory used by Synergy,
   it is usually in the form of "./yourdesignname_something.syn.run#",
   all that is necessary to do is to change the "#" to reflect the latest
   run directory
10. save the file and quit the editor
11. xnf_out
    it will create a "*.xnf" file with the same name as the design name but,
    due to some mysterious reasons, in UPPER case. This file located
    in the "xilinx" run directory, specified in the "xnf_out" file, MUST
    be renamed to lower case.
12. for user pin assignments, edit the file "filename.cst" that
    is located in the "xilinx" run directory.
13. picxilinx &
14. "Setup" will bring up a pop-up window titled "Xilinx"
    under "GLOBAL OPTIONS" specify all but
        leave the "Package File" as "default"
    under "NETLIST OPTIONS" select
        Import Netlist
        Generate Constraints File Template
    under "PHYSICAL OPTIONS" select
        Verilog Stand Alone
        User Pins Only
15. P & R
16. Physical
    it generates "*.v" and "*.sdf" files
17. generate VHDL shells for the verilog files just created:
    verilog +vhdl_crshell filename.v
18. compile the new VHDL file and modify the test bench for post-synthesis
    simulation, i.e., simulation with back-annotation results.
19. quit "HDL Desktop"

This procedure is put together to help new users to get thru Cadence tools and
to speed up the initial learning curve.

Last Modified on: 10-28-96
Cadence Version: 9502
Mahyar Malekpour

Procedure for using the many files and tools of Cadence development
environment to synthesize and implement a given design in VHDL:

1. Change directory to your working directory.
   cd YourWorkDirectory

   Note: "YourWorkDirectory" is the directory where your VHDL codes are.

2. Invoke "HDL Desktop" to run Leapfrog and Synergy:
   hdldesk &

From within hdldesk:

3. Compile all VHDL codes using hdldesk

4. Select the architecture to be synthesized, this will enable the "Synthesize" button

5. Run "SynrgCheck" from the hdldesk for a quick synthesizability check, or
   run "Synthesize" for a full synthesis of your architecture

   Note: "SynrgCheck" will operate from within hdldesk, while
   "Synthesize" will invoke Synergy.

From within Synergy:

6. Select the target library; 3000, 4000, or etc.

7. Select "Run Synthesizer ...",
   it will bring up a pop-up window with the caption bar "Run Synthesizer and Optimizer"

7.1. Select Generate Schematic option, if you desire to see the schematic
   Note: the "type" should be set to "Composer"

7.2. Set Constraint Priority to "cost" (a must for XC3000)

7.3. Set Job Priority to "Highest" which is numercial zero

7.4. Select "STD_LOGIC", if it is not selected

7.5. "OK" it. The pop-up window will disappear and the synthesis will begin

8. MUST wait for the synthesis to finish.

   Note: DO NOT hit any key or buttons until the synthesis is finished.
   Depending on the size of your design, it will take from one to a few
   minutes for the Synergy to finish synthesizing your code.

   Be patient!

9. To view the synthesis results, from the menu bar, select
   Show --> Output --> Composer Schematic
   It will bring up two the composer related windows

   Note: The Composer schematic viewer is VERY PRIMITIVE and with very few functions.
   You can zoom in and out, pan left and right, and plot the schematic.
   I use it for plotting and visual verification of the synthesis results.

   Note: DO NOT attempt to modify and save the modified schematic!

10. Quit schematic viewer!

11. Quit Synergy!

From the Unix environment:

12. Edit "xnf_out" file and specify the latest run directory used by Synergy,
   it is usually in the form of "./YourDesignName_Something.syn.run#",
   all that is necessary to do is to change the "#" to reflect the latest
   run directory

13. Save the file and quit the editor

```
      If you don't have a "xnf_out" file, then create it.  Note you only need to create it once.
      Here is a sample of a typical xnf_out file:

      xnfout -lib Opt -addio -rundir YourXilinxDirectory -spath
"./YourDesignName_Something.syn.run#
      /usr/local/cds-9502/share/library/xilinx/cds /usr/local/cds-9502//tools/dfII/etc/cdslib"
YourDesignNameinCaps

      Legend:
      YourXilinxDirectory            is the directory where the synthesis results will be.
                                       You need to create this directory once and prior to
                                       running xnf_out command.

      YourDesignName_Something.syn.run#      is the directory where all the temporary files will
be.
                                       Synergy creates a new directory after every run.

      YourDesignNameinCaps           is your design name, i.e., your design entity name,
                                     and not necessarily the file name of your design in
                                     caps (UPPER CASE).


      Note: The xnf_out must be an executable file.  Here is the Unix command to make
            this file executable:
                    chmod 744 xnf_out
            Note that it has to be done only once and after creating the "xnf_out" file.

14.   xnf_out
      It will create a "*.xnf" file with the same name as the design name and place it
      in the "YourXilinxDirectory" directory, specified in the "xnf_out" file.

15.   View the "xnfout.log" file for possible errors

16.   For user pin assignments,
      edit the file "filename.cst" that is located in the "YourXilinxDirectory" directory.
      If there is no such file there, then create one.

      Note: "*.cst" file format for the Xilinx xc3000 series is as follows:
      place block Your_I/O_Pin_Name P#;
        Example:
        place block ADDRESS<7> P2;

      Note: "*.cst" file format for the Xilinx xc4000 series is as follows:
      place instance Your_I/O_Pin_Name : #;
        Example:
        place instance ADDRESS<7> : C5;

      Note: For further details please see the Xilinx manuals.


There are two ways of generating hex files for programming the Xilinx chips:
a.  via the Xilinx front end tool called "picxilinx" or
b.  creating a make file and running it at the command line.

Option a, via "picxilinx" and from "YourWorkDirectory":

17.   Run Pic-Xilinx program:
      picxilinx &

17.1. "Setup" will bring up a pop-up window titled "Xilinx"
      under "GLOBAL OPTIONS" specify
          "Design Name" to "design-name",
          "Work File" to "./file-name.wrk",
          "Run Directory" to "YourXilinxDirectory,
          "Part Name" to "XC3020PC84" or "4005APG156" or other Xilinx parts, and
          leave the "Package File" as "default"
      under "NETLIST OPTIONS" select
          "Import Netlist"
          "Generate Constraints File Template"
      under "PHYSICAL OPTIONS" select
          "Verilog Stand Alone"
          "User Pins Only"

17.2. Select "P & R"
```

```
      Note: DO NOT hit any key or buttons until it is finished.
            Depending on the size of your design, it will take from one to a few
            minutes for the Place & Route to finish.

            Be patient!

17.3. Select "Physical"
      It generates "*.v" and "*.sdf" files that are used in backannotation
      and post-synthesis simulations.

17.4. Generate VHDL shells for the verilog files just created:
      verilog +vhdl_crshell filename.v

17.5. Compile the new VHDL file and modify the test bench for post-synthesis
      simulation, i.e., simulation with back-annotation results.

17.6. Quit "HDL Desktop"


Option b, via make files and from your "YourXilinxDirectory" directory:

18.   xmake FileName.mak

      Note: Depending on the size of your design, it will take from one to a few
            minutes for the Place and Route to finish.

            Be patient!

      Note: You need to create a make file for your design and target chip only
            once.  This file doesn't need to be modified there after for that design.


      Here is a sample make file for programming Xilinx 3000 series:

      #
      # Created by XMAKE Version 5.0.0 on Tue Jan 16 14:48:52 1996
      #
      # The following options were used: -P 3020PC84-100 -X
      #
      # The following is the hierarchy of the design 'FileName.xnf'
      #
      DEFAULT_TARGET FileName.bit

      FileName.bit : FileName.lca
            makebits -S0 -R2 -XB -YA FileName.lca

      FileName.lca : FileName.map
            map2lca -P 3020PC84-100 FileName.map FileName.lca
            apr -W -Y FileName.lca FileName.lca -c FileName.cst

      FileName.map : FileName.xtf
            xnfmap -P 3020PC84-100 FileName.xtf FileName.map

      FileName.xtf : FileName.xff
            xnfprep FileName.xff FileName.xtf parttype=3020PC84-100 cstfile=FileName.cst

      FileName.xff : ld.xnf FileName.xnf
            xnfmerge -A -D xnf -P 3020PC84-100 FileName.xnf FileName.xff



      Here is a sample make file for programming Xilinx 4000 series:

      #
      # Created by XMAKE Version 5.1.0 on Thu Apr  4 13:55:08 1996
      #
      # The following options were used: -P 4005APG156-5 -X
      #
      # The following is the hierarchy of the design 'FileName.xnf'
      #
      DEFAULT_TARGET FileName.bit

      FileName.bit : FileName.lca
            makebits FileName.lca

      FileName.lca : FileName.cst FileName.xtf
```

```
        ppr FileName.xtf -run_pic2map parttype=4005APG156-5
        xdelay -D -W FileName.lca

    FileName.xtf : FileName.xff
        xnfprep FileName.xff FileName.xtf parttype=4005APG156-5 cstfile=FileName.cst

    FileName.xff : ld.xnf fdp.xnf FileName.xnf
        xnfmerge -A -D xnf -P 4005APG156-5 FileName.xnf FileName.xff
```

To program an FPGA via an EPROM, the FPGA bit file needs to be converted to an EPROM hex file by the following unix command and from your "YourXilinxDirectory" directory:

19. `makeprom -f mcs -u StartAddress FileName.bit`

    Legend:

StartAddress	is the start address of the EPROM where the hex file will be loaded; typically, 100.
FileName.bit	is the FPGA bit file name that is in "YourXilinxDirectory" directory.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	April 2002	Technical Memorandum

4. TITLE AND SUBTITLE
Fly-By-Light/Power-By-Wire Fault-Tolerant Fiber-Optic Backplane

5. FUNDING NUMBERS
WU 728-30-10-03

6. AUTHOR(S)
Mahyar R. Malekpour

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
NASA Langley Research Center
Hampton, VA 23681-2199

8. PERFORMING ORGANIZATION REPORT NUMBER
L-18158

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
National Aeronautics and Space Administration
Washington, DC 20546-0001

10. SPONSORING/MONITORING AGENCY REPORT NUMBER
NASA/TM-2002-211632

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION/AVAILABILITY STATEMENT
Unclassified–Unlimited
Subject Category 60 Distribution: Nonstandard
Availability: NASA CASI (301) 621-0390

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*
The design and development of a fault-tolerant fiber-optic backplane to demonstrate feasibility of such architecture is presented. The simulation results of test cases on the backplane in the advent of induced faults are presented, and the fault recovery capability of the architecture is demonstrated. The architecture was designed, developed, and implemented using the Very High Speed Integrated Circuits (VHSIC) Hardware Description Language (VHDL). The architecture was synthesized and implemented in hardware using Field Programmable Gate Arrays (FPGA) on multiple prototype boards.

14. SUBJECT TERMS	15. NUMBER OF PAGES
Fly-By-Light/Power-By-Wire	117
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	UL